UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE GEOGRAFÍA E HISTORIA

MÁSTER EN TECNOLOGÍAS DE LA INFORMACIÓN GEOGRÁFICA

TRABAJO FIN DE MÁSTER
CURSO 2017-2018

Clasificación automática, visualización y análisis de fotografías del camino de Santiago. Obtención de conclusiones sobre los hábitos fotográficos de los peregrinos.

Automatic classification, visualization and analysis of photographs of the Camino de Santiago. Extraction conclusions about the photographic habits of travelers.

Rubí Rincón Peña

Junio 2018

Tutores: Borja Moya Gómez y Juan Carlos García Palomares

Resumen

El propósito general del proyecto será analizar un conjunto de fotografías geolocalizadas obtenidas de la red social Flickr y todas ellas tomadas en el camino de Santiago español.
A partir de esta información analizaremos el contenido visual de cada fotografía utilizando la interface de programación de aplicaciones (API) proporcionada por Google en la herramienta Cloud Vision.

Esta herramienta permite establecer una serie de parámetros de entrada definidos para obtener una clasificación de tres etiquetas (tag1, tag2 y tag3), una mejor clasificación (best tag), y tres urls de fotos relacionadas con la original. Con esta información agruparemos las fotografías en grupos según su clasificación en etiquetas. El objetivo es alimentar un análisis espacial, utilizando un Sistema de Información Geográfica, de las temáticas identificadas en cada una de las fotografías.

La utilización de las herramientas proporcionadas por Google Cloud Vision se aplicarán en el proyecto a través de procesos programados en el lenguaje de programación Python que nos permitirán automatizar la clasificación de nuevas fotografías de forma rápida y sencilla.

Para una mejor comprensión de los datos obtenidos crearemos un mapa web, disponible en la red de forma pública. Para ello utilizaremos la infraestructura de red que nos proporciona Amazon a través de sus servicios web AWS (Amazon Web Services) como EC2 (máquinas físicas), S3 (almacenamiento de contenido), DynamoDB (base de datos) y Lambda (procesos programados).

Los datos se mostrarán en un mapa web que podrá ser "alimentado" con valores de fotografías nuevas a partir de su id en Flickr y mostradas en el mapa de forma automática. Esto permite
mostrar las fotos según su localización por categorías y analizar los datos en función de su importancia.

Para testar la herramienta y presentar un ejemplo de su posible aplicación se ha usado una zona del Camino de Santiago concretamente en la provincia de Logroño y más localizada en su mayoría en Santo Domingo de la calzada. De esta manera podemos analizar las clasificaciones obtenidas para valorar si la aplicación del proceso con conjuntos de valores más grandes podría ser válido y aplicable a todo el camino de Santiago.

Palabras clave

clasificación automática, reconocimiento visual, fotografía, visualización espacial
automatic sorting, visual recognition, photography, spatial visualization

Índice de contenidos

1.- Introducción....4

1.1.- Objetivos del trabajo 4

1.2.- Estructura del Trabajo....5

2. Herramientas utilizadas....7

2.1 API Flickr (https://www.flickr.com/services/api/)....7

2.2.- API Google Vision....8

2.3.- API Amazon AWS....10

2.4.- Herramientas de renderización de datos geográficos, leatlef.js, mapbox....16

2.5.- Uso de ArcGIS....17

3.- Datos y tratamientos previos 18

3.1.- Datos iniciales e incorporación de coordenadas geográfica....18

3.2- Limpieza de datos 21

3.3. - Incorporación de las URLs y photos....22

4.- Google Cloud Vision para clasificación automática de imágenes....25

5.- Procesos batch para el guardado de datos, AWS infraestructura....34

6.- Análisis y primeras visualizaciones de datos sobre un mapa base. Renderización en Java Script con leaflet....36

7. - Demo y conclusiones....44

Bibliografía y páginas web de referencia....45

1.- Introducción

En este trabajo queremos establecer un procedimiento automático para la clasificación del contenido y la visualización geolocalizada de las fotografías tomadas a lo largo del camino de Santiago. Más concretamente analizaremos el contenido de 2500 fotografías realizadas en la provincia de Logroño todas ellas relacionadas con el camino de Santiago.

Primeramente analizaremos los pasos a seguir para llegar a los resultados esperados. Para ello estableceremos los objetivos deseables alcanzables a continuación.

1.1.- Objetivos del trabajo

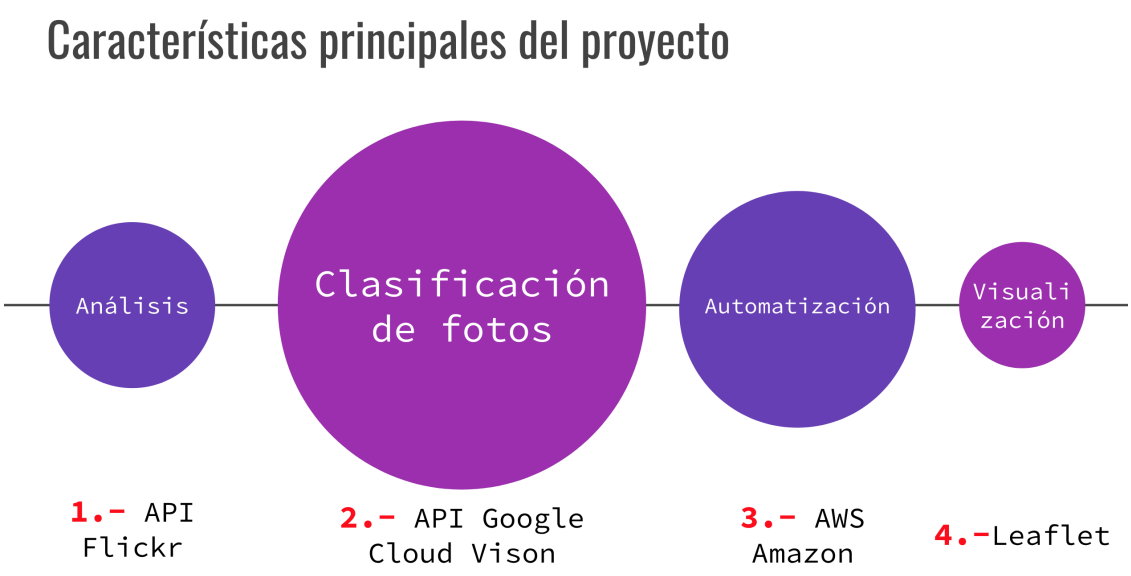
El objetivo principal del trabajo es tener un sistema automatizado para poder clasificar y visualizar fotografías geolocalizadas obtenidas de la red de Flickr. Trabajaremos siempre con fotografías públicas en las que no habrá problema con su manejo y análisis.

Para cumplir con el objetivo principal se proponen los siguientes objetivos específicos:

- Manejar de forma ágil las interfaces relacionadas con datos geográficos que nos proporciona Flickr, Google y Amazon. Estudiar su API (interfaz de programación de aplicaciones) de usuario públicas para aplicarlas en scripts programados en el lenguaje Python que realizarán las tareas de clasificación y de obtención de datos de entrada que tenemos que enviar a Google Cloud Vision.
- Analizar la API de la red social Flickr para poder descargar y trabajar con fotografías geolocalizadas de esta red social. Necesario para poder enviar también a Google Cloud Vision, ya que la fuente principal de la clasificación será el fichero físico de la fotografía o en su lugar una url de acceso público de la misma.
- Ser capaz de manejar la API Google Vision para clasificar fotografías geolocalizadas según la temática de la foto. Incluirá el análisis de los parámetros de entrada que necesitará el API por cada una de nuestras fotografías. Serán ficheros con extensión json por cada una de las fotografías con un formato fijo pero con valores diferentes para cada una de ellas.
- Utilizar y manejar el API Amazon AWS para poder incluir un mapa web con todos los datos obtenidos que serán almacenados en una base de datos de tipo no relacional dynamoDB que nos proporciona Amazon, con la que interaccionaremos desde ficheros javascript almacenados en otro de los servicios web que nos facilita Amazon como es S3 (almacenamiento seguro en la nube) donde también almacenaremos el fichero index.html de nuestra página web junto a los recursos necesarios como ficheros de estilo css, imágenes, y librerías de terceros como leaflet o mapbox. Todos ellos serán parte de la web con nuestros resultados, donde se podrán ver, analizar y descargar los datos resultantes.
- Manejar Herramientas de renderización de datos geográficos para mostrar cada una de las fotografías clasificadas en nuestro mapa junto a las fichas por cada una de

ellas donde mostraremos las etiquetas de clasificación junto con el score o puntuación de acercamiento al contenido analizado. Es decir, por cada etiqueta tendremos un dato entre 0 y 1 que corresponderá con la precisión que el proceso ha podido obtener entre la forma reconocida en la fotografía y la forma real que existe en la fotografía. Siendo uno el valor con mayor precisión. En estas fichas también mostraremos la etiqueta con mayor score obtenido, y tres fotografías con contenido relacionado a la original, que podrán ser visualizadas en su url origen de la que ha sido obtenida mediante un link.

- Implementar los resultado en un SIG (en este caso ArcGIS y carto) para poder realizar análisis espaciales y obtener rutinas entre las fotografías tomadas por los usuarios.
- Implementar un visor web de datos geolocalizados para visualizar de forma clara los datos obtenidos y poder hacerlos públicos en una web.



.- Esquema general con los objetivos del proyectos y tecnologías principales utilizadas.

1.2.- Estructura del Trabajo

1. Análisis de datos iniciales

   Filtraremos los datos para comprobar la validez de todos ellos. Comprobamos que todas las fotografías siguen existiendo en la plataforma original, eliminaremos aquellas que ya no existan en la red social Flickr para evitar errores en los procedimientos automáticos posteriores.

2. Planteamiento técnico sobre arquitectura y herramientas a utilizar.

Analizaremos que tipo de datos tenemos y donde necesitamos almacenarlos para poder luego consultarlos. Será necesario tener una base de datos donde a partir de los datos de nuestro shapefile inicial podamos extraerlo a una tabla no relacional que podamos consultar mediante claves de acceso, por ejemplo un identificador único por fotografía. Para ello crearemos en al base de datos dynamoDB, proporcionada por Amazon, una tabla con nombre DATA_IMAGE_SANTIAGO donde almacenaremos la información actual y posteriormente completaremos con datos como la url descargable de cada fotografía y su clasificación completa (campo por cada registro con nombre clasification_google_vision) y que será el resultado de llamar al API de Google Cloud Vision.

3. Obtención de resultados geojson

   Para la obtención de la clasificación de cada una de nuestras fotografías necesitaremos crear un fichero .json de entrada por cada una de ellas y obtendremos como resultado otro fichero .json con los datos del análisis del contenido de cada una de ellas. Para ello crearemos dos procesos programados en el lenguaje de programación Python. Uno de ellos guardará un fichero con estructura de nombre, idFotografiaEnFlickr.json con el contenido necesario por Google Cloud Vision. Este fichero será explicado y analizado más adelante. El resultado de la llamada se guardará en un campo más de la tabla DATA_IMAGE_SANTIAGO con nombre clasification_google_vision. Estos procesos se ejecutarán como funciones lambda que no dejan de ser código Python que puede ser ejecutado de forma automática al suceder una acción. En nuestro caso será el guardar un fichero en S3 con la estructura de entrada necesaria para el proceso de Google Cloud Vision. O subir un fichero .csv al S3 con todos los datos iniciales de carga de nuestra tabla DATA_IMAGE_SANTIAGO.

4. Visualización de datos

   Para la visualización de datos accederemos a los datos de nuestra tabla en dynamoDB DATA_IMAGE_SANTIAGO mediante código javascript que podrá ser ejecutado desde un navegador web. Utilizaremos librerías como leaflet para tratar los datos y mostrarlos en un mapa basado en bases de mapbox. Todo ello se almacenará en un sistema de almacenamiento de ficheros seguro en la nube y accesibles desde la web como es S3 proporcionado por AWS (Amazon Web Services).

5. Análisis

   Utilizaremos la librería leaflet para poder mostrar capas (layers) de información con las fotografías clasificadas por su etiqueta con mayor score. Mediante código javascript generaremos cada uno de los ficheros .geojson con la información de todas las fotografías clasificadas por cada una de las etiquetas obtenidas. Por ejemplo podremos cargar en nuestro mapa resultados todas las fotografías clasificadas con la tag “sculture” extraídas del total de los datos junto con un mapa

de calor donde se podrán visualizar las zonas con mayor concentración de fotografías tomadas en relación a esa etiqueta. Estos ficheros también podrán ser descargados fisicamente en nuestro ordenador para uso libre.

6. Propuestas finales de mejoras

Platearemos como conclusiones mejoras al proyecto con mejoras en los procesos automáticos y la carga de mayor cantidad de datos para concluir que el proceso podría ser aplicado a mayores cantidades de datos.

2. Herramientas utilizadas

Para llegar al objetivo del proyecto utilizaremos procesamientos desarrollados en Python y tendrán como base las API (Application Programming Interface) de las principales plataformas de que haremos uso, Flickr, Google Vision, AWS y leatlef. Paralelamente utilizaremos también ArcGis para analizar de forma estática los datos obtenidos.

A continuación describimos de forma resumida cada una de las APIs y su objetivo principal de su uso en nuestro proyecto:

2.1 API Flickr (https://www.flickr.com/services/api/)

Flickr es una de la redes para compartir fotografías entre usuarios más grandes en internet. Nos proporcionan información por cada foto muy valiosa, como metadatos, etiquetas, geolocalización y datos exif. Es más, a partir de su API nos proporciona a los usuarios la forma de acceder a estás fotografías y toda su información asociada. Es totalmente gratuito y accesible. A partir de aquí, en nuestro proyectos nos haremos una cuenta de desarrollador para poder tener nuestra clave de acceso al API y poder interactuar con la red de Flickr para obtener información de nuestra base de fotografías (2500) de las que partimos y con las que trabajaremos.

Para adquirir nuestra clave personal para poder autentificarnos en flickr, lo haremos accediendo a la siguiente url:

https://www.flickr.com/services/apps/create/

Toda la documentación del API está accesible desde la siguente url:

https://www.flickr.com/services/api/

Extraeremos procedimientos algorítmicos para poder enriquecer nuestra base de fotografías con fotografías nuevas basándonos en puntos de acceso de la API de Flickr, lo que llamaremos API methods.

Dentro de todos los API Methods que nos ofrecen nosotros trabajaremos con los especificados a continuación:

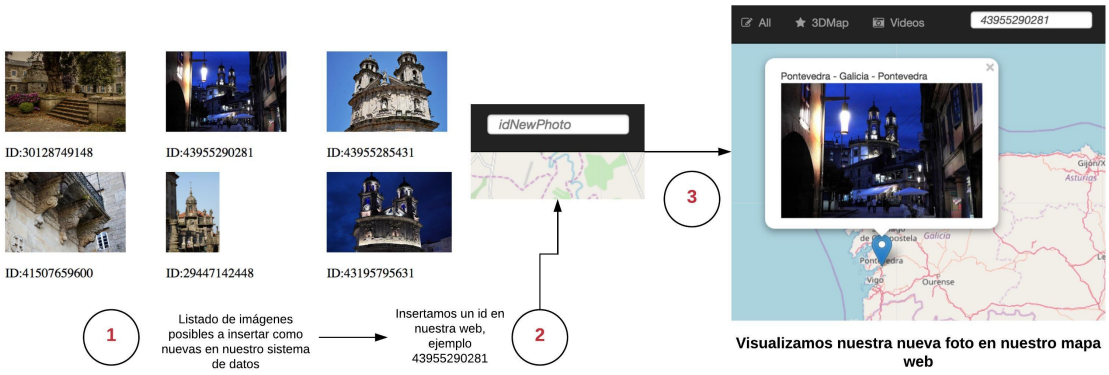
flickr.photos.getSizes

Devuelve los tamaños disponibles para una foto. El usuario que llama debe tener permiso para ver la foto y lo usaremos para poder obtener la url que nos servirá de base para la entrada de datos que necesita Google Vision para poder analizar y clasificar automáticamente nuestras fotos.

https://www.flickr.com/services/api/flickr.photos.getSizes.html

2.2.- API Google Vision

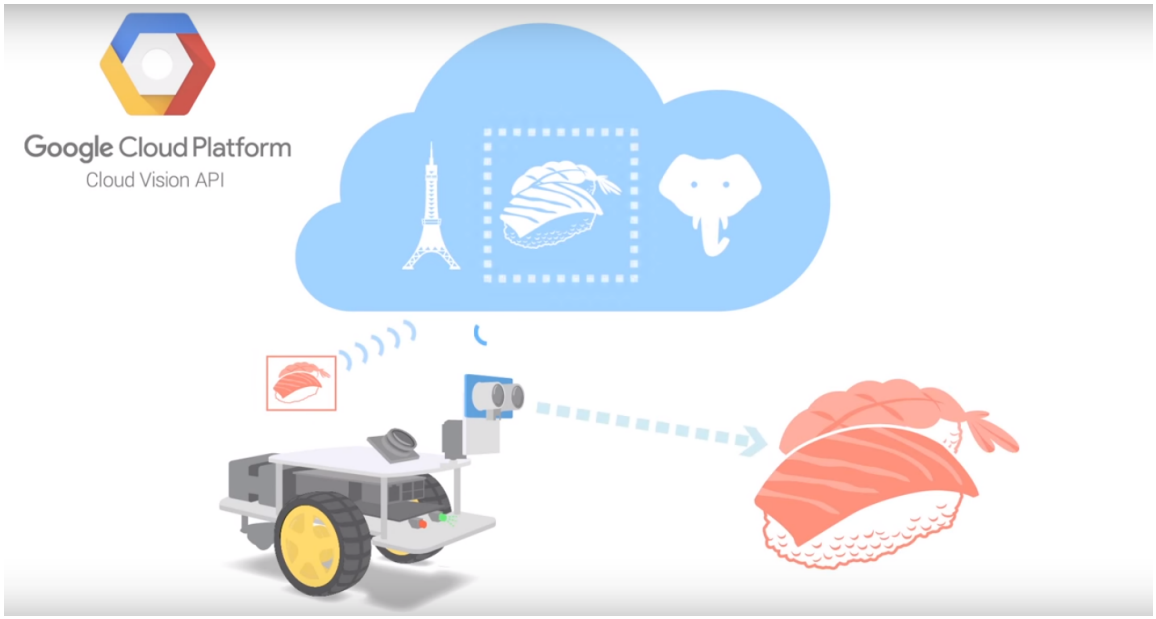
Cloud Vision API es una herramienta que nos proporciona Google con la que podemos analizar, extraer y comprender el contenido de las imágenes a analizar. En nuestro proyecto será la herramienta principal. A través de su API, de forma gratuita, construiremos nuestro proceso automático para poder analizar primeramente nuestras 2500 fotos (carga inicial de datos) y posteriormente establecer un proceso que alimente y clasifique de forma automática nuevas fotos que podrán ser proporcionadas a través de nuestra web por usuarios de internet o por nosotros mismos a partir solamente del identificador de la foto que nos interese añadir al conjunto de datos del proyecto.



.- Proceso en pasos para añadir una nueva foto al proyecto.

Para nuestro proyecto usaremos la versión gratuita que Google nos proporciona, pero para un mayor potencial podría usarse una versión de pago. Referencias sobre los precios sobre el uso de la herramienta podemos consultar la siguiente página web:

https://cloud.google.com/vision/pricing



.- Esquema del funcionamiento de Google Vision.

Vision API proporciona una interfaz RESTful que facilita la tarea de tener que desarrollar algoritmos de procesamiento de imágenes. Vision API soporta formatos de archivo de imagen como JPEG, BMP, RAW, PNG. Para un análisis lo más preciso posible, la resolución de imagen mínima recomendada es VGA (640 × 480 píxeles), y el tamaño de los ficheros no debe superar los 4 MB.

Nosotros aplicaremos el reconocimiento del contenido a través de llamandas curl a su API. Para más información podemos consultar la documentación facilitada por Google en su web: https://cloud.google.com/vision/docs/using-curl

Previamente necesitaremos:

1. Instalar la herramienta curl (https://curl.haxx.se/download.html)

Curl es una herramienta de línea de comandos para realizar solicitudes de URL del lado del cliente. Está disponible para la mayoría de las plataformas, incluidos Linux, Windows y macOS.

2. Crear la request Json y guardarla en un fichero con extensión .json.
3. Enviar la petición request, deberemos especificar nuestra clave de acceso quenos identifica como usuarios dados de alta, Google nos permitirá el acceso a sus servicios a partir de nuestro logado.
4. Recibir y tratar la respuesta. La información se recibe en formato json, la información que nos interesará estará contenida en el campo “webentities”.

La herramienta curl estará instalada en nuestro entorno amazon EC2, con lo cual podremos ejecutar un proceso que haga está llamada a Cloud Vision. Necesitaremos establecer un

proceso para generar de forma automática los ficheros .json que Google Vision espera recibir como entrada junto con la llamada curl y por último tendremos que leer y tratar la información que recibamos como respuesta a la llamada curl por cada una de nuestras fotografías. Crearemos un proceso que lea el campo "webentities" y almacenaremos la información de clasificación en un campo de una base de datos en Amazon DynamoDB, así podremos tener almacenada toda la información de nuestro sistema de datos. Podrá ser consultado tantas veces como queramos.

La lectura de todos los datos la realizaremos desde nuestra web ejecutando una llamada Ajax desde nuestra web haciendo uso de código Javascript y renderizaremos la información ya almacenada en nuestra base de datos dentro de un mapa accesible y público desde internet.

2.3.- API Amazon AWS

A la hora de automatizar y obtener una visualización de los análisis de nuestros datos iniciales nos planteamos que estructura y arquitectura técnica puede darnos la visibilidad suficiente para poder visualizar todos nuestros resultados.

Para ello haremos uso de los servicios web proporcionados en el perfil gratuito de Amazon. El listado y objetivo de cada uno de ellos que utilizaremos en nuestro proyecto son los siguientes:

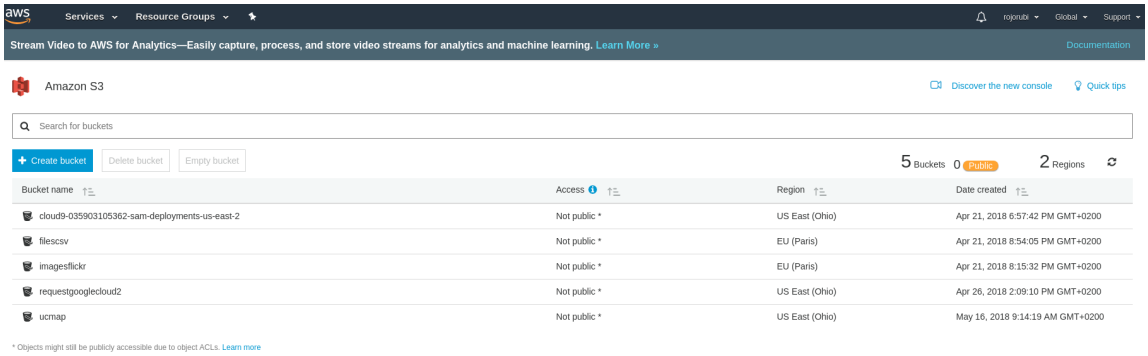
S3

Amazon Simple Storage Service es un servicio de almacenamiento, similar a un disco duro en la nube de Amazon, diseñado con la finalidad de facilitar el escalado basado en web, es decir, si necesitamos más espacio Amazon nos lo proporciona.

Tendremos disponible una interface de servicios web para que de una manera sencilla podamos almacenar y recuperar nuestros datos en cualquier momento desde cualquier parte de la web.

Nos facilitará mucho el trabajo a la hora de hacer pública nuestra web. Subiremos a nuestro S3 todos los ficheros referidos a la parte cliente de nuestra web. Es decir, todos los ficheros .html, .js (extensión javascript), imágenes, ficheros de estilo css.

Amazon nos ofreces una infraestructura para un uso y mantenimiento perfecto que nos ahorrará tiempo y nos proporcionará un uso fiable, seguro y rápido.



.- Screenshot de la pantalla de configuración del servicio web S3 en AWS.

DynamoDB

Amazon DynamoDB es un servicio de bases de datos NoSQL, es decir no se utiliza el lenguaje SQL como lenguaje de consulta y lo más importante, los datos no necesitan estrucutras fijas como tablas, no admiten operaciones join entre tablas, entre otras operaciones.

Amazon nos proporciona un sistema administrado, nos permite abstraernos de las tareas de mantenimiento, como la configuración de hardware, instalaciones de software o administrativas. Además la escalabilidad vuelve a ser proporcionada de manera óptima.

Con DynamoDB, podemos crear tablas de base de datos capaces de almacenar y recuperar cualquier cantidad de datos, pudiendo atender cualquier nivel de tráfico de solicitudes.

Para nuestro proyecto crearemos una tabla en la que guardaremos la información por cada una de la fotografías de nuestro sistemas, una vez obteniendo la respuesta con Google vision completaremos el registro del identificador de nuestra foto con la información de las "webentities" que Google Vision nos proporciona.

La tabla la llamaremos "**DATA_IMAGE_SANTIAGO**" y cada uno de sus registros contendrá los campos:

_id: identificador único de cada fotografía.

clasificacion_google_vision: respuesta de Google Cloud Vision con la clasificación que nos indicará el contenido reconocido en nuestra fotografía.

dates_take: fecha en la que fue tomada la fotografía.

id: identificador necesario en Dynamodb

latitude: latitud donde fue tomada la fotografía.

longitude: longitud donde fue tomada la fotografía.

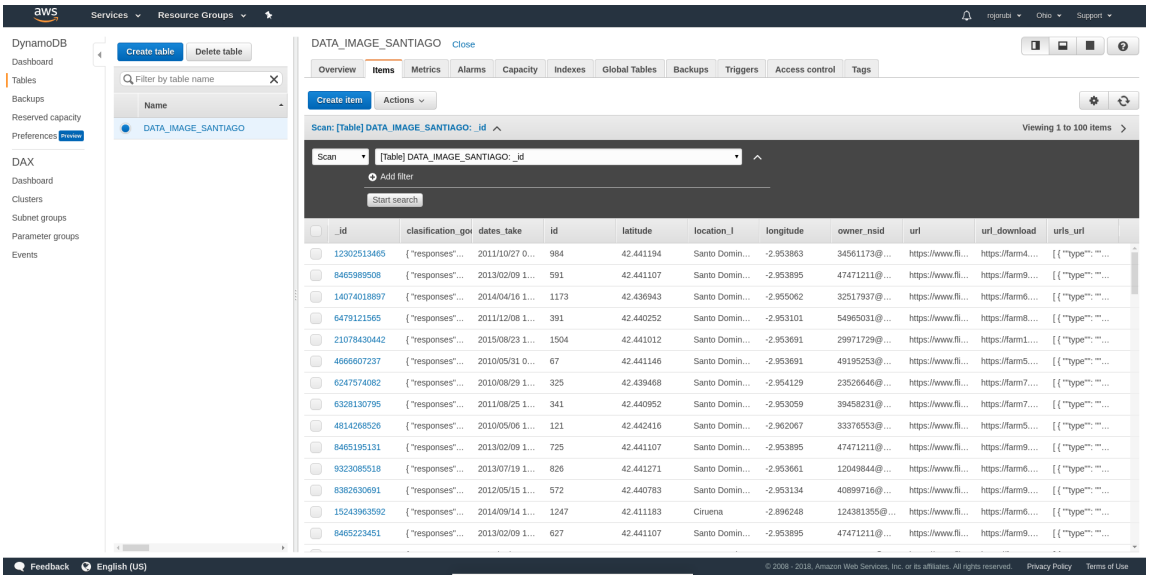
location_l: nombre del lugar donde fue tomada la fotografía.

owner_nsid: usuario en Flickr que tomó la fotografía.

url: url de la fotografía en Flickr

url_download: url pública que nos facilitará la descarga de la fotografía.

urls_url: más información sobre las urls de la fotografía.



.- Screenshot de la pantalla de configuración del servicio web DynamoDB en AWS.

EC2

Es un servicio nuevamente proporcionado por Amazon gratuito que nos permite tener capacidad de computación en la nube. Es decir, nos proporciona acceso al hardware necesario para el desarrollo de cualquier proyecto de software, tanto un proyecto pequeño como puede ser el nuestro y para el que haremos uso solo de una máquina EC2 con una sola instancia (entorno informático virtualizado) en Estados Unidos pero que manejaremos y administraremos desde cualquier ordenador conectado a Internet.

La configuración de la red, la seguridad y la administración del almacenamiento estará preestablecida y no tendremos que hacer nada nosotros, permitiéndonos tener tiempos de desarrollo mucho más cortos. También tendremos capacidad de autoescalado por si el tráfico de nuestra aplicación aumenta repentinamente o simplemente las necesidades del proyecto crecen (elestic compute cloud EC2).

Para más información hacemos referencia a la guía de documentación que nos proporciona Amazon en su web,

https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html

Enumeramos a continuación las características que EC2 nos proporciona y que consideramos como las más importantes para nuestro proyecto:

1. Entornos informáticos virtuales, conocidos como instancias.
2. Plantillas pre-configuradas para las instancias, o imágenes (AMI), despreocupándonos de la instalación del software base, como puede ser el sistema operativo de la máquina y el software adicional.
3. Manejo de claves pública y privada para mantener el sistema con seguridad.
4. Zonas de disponibilidad o regiones donde podemos tener los recursos disponibles. En nuestro caso los servicios los tenemos distribuidos en la región US East (Ohio).

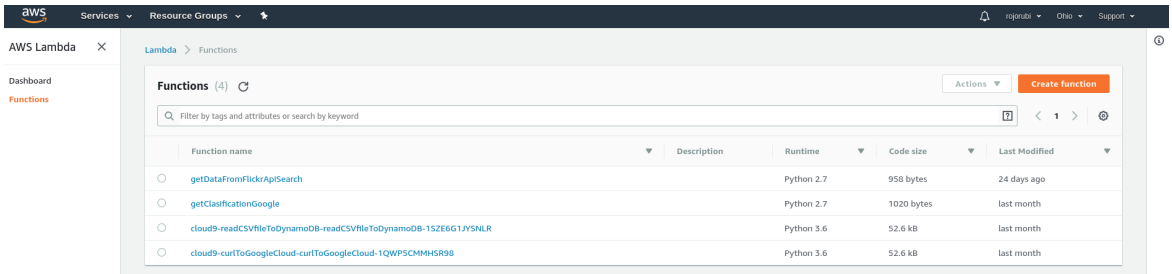
Lambdas

Con el servicio de Amazon lambdas podremos ejecutar código sin necesidad de administrar servidoes. El código, en nuestro caso escrito en lenguaje python se ejecutará solamente cuando sea necesario, es decir, podremos asociar a nuestro código eventos de ejecución. Concretamente nuestra lambda llamada *"getClasificationGoogle"* que es un fichero que contiene código python getClasificacionGoogle.py se ejecutará al suceder el evento "put en S3", es decir cuando se guarde un fichero en el S3 de nuestro proyecto se ejecutará el código de nuestra lambda.

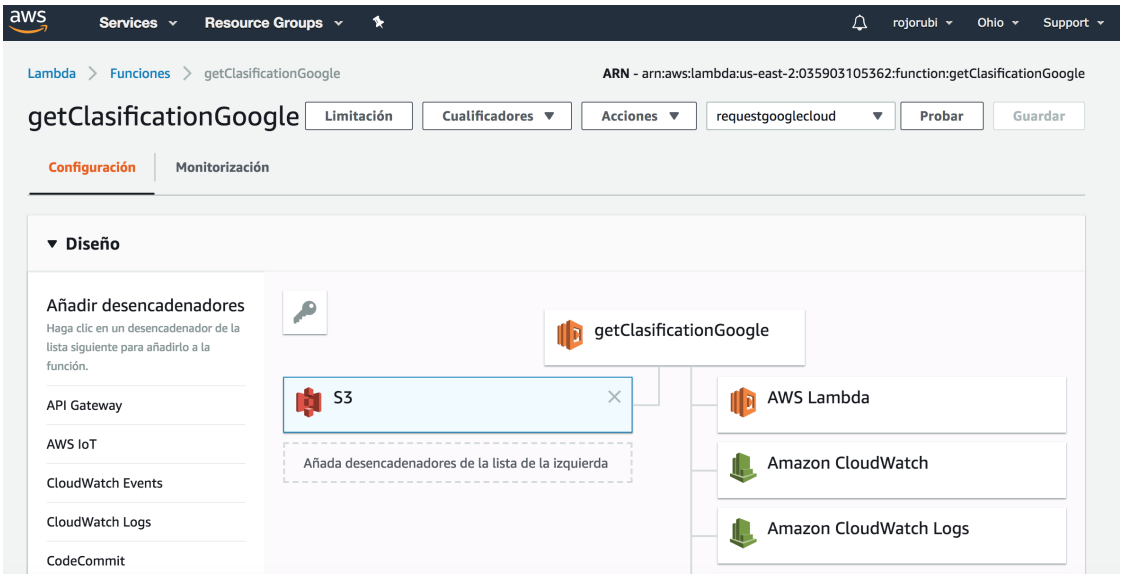
Desde nuestro proyecto, crearemos dinámicamente un fichero .json con la request que necesitamos enviar a través de la llamada curl a Google vision, este fichero lo generaremos y lo guardaremos en el S3, cuando sea guardado se ejecutará el código de nuestra lambda y será la lambda la encargada de realizar la llamada curl a Google Vision y a su vez de completar los datos del registro en la tabla de DynamoDB.

Para más información hacemos referencia a la documentación que facilita Amazon en su web:

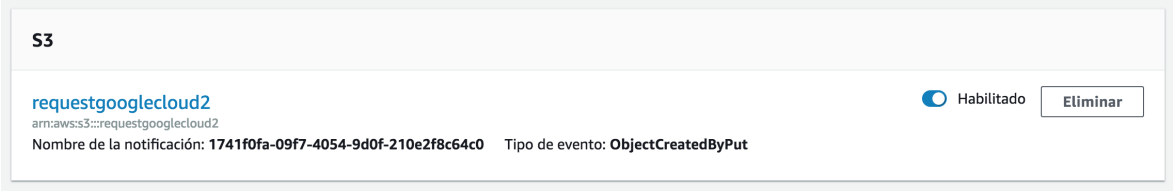
https://aws.amazon.com/es/lambda/features/



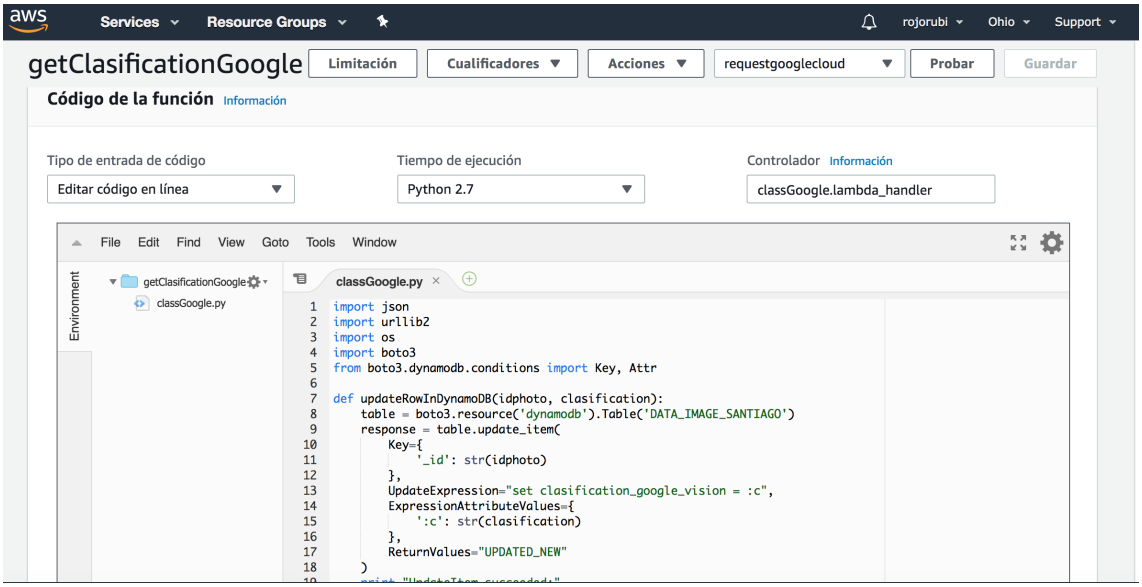
.- Screenshot de la pantalla de configuración de nuestro servicio web Lambda en AWS.



.- Captura de la carga de nuestra lambda en el entorno cloud que nos proporciona Amazon.



.- Configuración del evento que hace ejecutarse el código de nuestra lambda getClasificationGoogle.



.-Captura de la carga del código python contenido en nuestra lambda getClasificationGoogle.

```python
import json
import urllib2
import os
import boto3
from boto3.dynamodb.conditions import Key, Attr

def updateRowInDynamoDB(idphoto, clasification):
    table = boto3.resource('dynamodb').Table('DATA_IMAGE_SANTIAGO')
    response = table.update_item(
        Key={
            '_id': str(idphoto)
        },
        UpdateExpression="set clasification_google_vision = :c",
        ExpressionAttributeValues={
            ':c': str(clasification)
        },
        ReturnValues="UPDATED_NEW"
    )
    print "UpdateItem succeeded:"
    print(json.dumps(response, indent=4))
    print "end process."

#https://cloud.google.com/vision/docs/reference/rest/v1/images/annotate#AnnotateImageRequest
def callingToGoogle(idphoto, url_process_lead, payload_json):
    print "Calling to google..."
    req = urllib2.Request(url_process_lead, payload_json)
    req.add_header("Content-type", "application/json")
    req.add_header("encoding", "base64")
    res = urllib2.urlopen(req)
    response = res.read()
    print "Response to GOOGLE CODE CLASIFICATION IMAGE:",response
    #buscar en la tabla de dynamo el _id de la foto con idphoto e insertar el valor del campo clasification_google_vision
    updateRowInDynamoDB(idphoto, response)

#https://github.com/gxx/aws-lambda-python/blob/master/ARTICLE.md
def lambda_handler(event, context):
    print "Event:",event
    # Obtain the bucket name and key for the event
    bucket_name = event['Records'][0]['s3']['bucket']['name']
    print "bucket_name:",bucket_name
    key_path = event['Records'][0]['s3']['object']['key']
    print "key_path:",key_path
    idphoto = key_path.split(".")[0]
    print "idphoto:",idphoto
    payload_json_s3 = boto3.resource('s3').Object(bucket_name, key_path).get()['Body'].read() # Retrieve the S3 Object
    print "body from s3 json:",payload_json_s3
    url_process_lead = 'https://vision.googleapis.com/v1/images:annotate?key=AIzaSyBAkCyuOZpyerrB-d2Fo4dm2Egw8dFm_SM'
    payload_json=json.dumps(json.loads(payload_json_s3))
    callingToGoogle(idphoto, url_process_lead, payload_json)
```

.-Código python completo y comentado de nuestra lambda getClasificationGoogle.

Amazon Cognito

Amazon cognito será el servicio de Amazon que utilizaremos en el proyecto para crear un grupo de usuarios que tendrán acceso a todas las utilidades que abstraeremos a nuestra web y que tendrán interacción con el conjunto de servicios web de Amazon que utilizamos, como la base de datos de DynamoDB (para escribir y completar la información de nuevas fotografías tanto procedente de Flickr, información básica de la foto y localización, como de Google Vision, clasificación del contenido de la fotografía) y el servicio de S3 (para poder subir y almacenar las fotos nuevas que incorporemos).

Para más información hacemos referencia a la documentación que proporciona Amazon en su web:

https://docs.aws.amazon.com/es_es/cognito/latest/developerguide/what-is-amazon-cognito.html

2.4.- Herramientas de renderización de datos geográficos, leatlef.js, mapbox.

Mapbox es una plataforma de código abierto para la creación de mapas enfocada principalmente a desarrolladores.Se basa en mapas vectoriales para diseñar y personalizar el estilo de los mapas.

Leaflet es una librería javascript para la creación de mapas interactivos. Nos permite utilizar los tiles del servicio que prefiramos para pintar el mapa. En el proyecto utilizaremos los tiles de OpenStreetMap.

Para poder cargar el mapa, necesitamos indicarle a leaflet el identificador un elemento div cargado en el DOM de nuestra página. Le indicaremos a leaflet que utilice el elemento con id "map" y añadiremos un "tileLayer" a nuestro mapa. Un "tile" es una imagen que representa un área determinada. Cada nivel de zoom cargará los "tiles" asociados a ese nivel de zoom, los cuales tendrán más o menos detalle en función de la escala del nivel de zoom.

Añadir marcadores a nuestro mapa para indicar cada una de nuestras fotografías junto con un popup con información de cada marcador, en nuestro caso de cada fotografía. Este popup será un html con la siguiente información:

Title, related images, longitude, lattitude, best clasification, tags (score), url flickr y photo.

Como necesitamos representar una cantidad de datos grande es posible que a un nivel de zoom bajo nuestro mapa se vea sobrecargado. Además el renderizar una gran cantidad de marcadores también es bastante costoso.

Para ello agruparemos los marcadores en clusters, de forma que los marcadores que se aglutinan en un área determinada se representarán como un único marcadores, y se separarán cuando se alcance un nivel de zoom determinado (o al hacer click, se cambiará el nivel de zoom). Para ello leaflet cuenta con un plugin llamado Leaflet.markercluster. Para instalarlo podemos utilizar npm (en sistema linux): npm install --save leaflet.markercluster

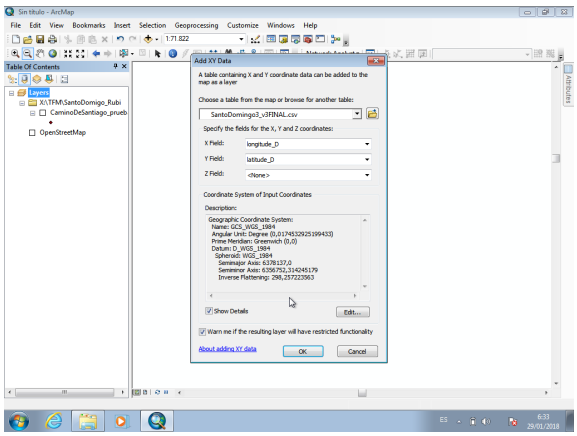
Como conclusión para la elección de leaflet diremos que Leaflet.js es una librería bastante completa. Es bastante intuitiva y su API es bastante simple. Además, nos permite personalizar muchos aspectos, como el icono de los marcadores o el comportamiento de los mismos. También dispone de bastantes plugins como en el caso de los clusters.

El hecho de que podamos utilizar cualquier servicio de mapas que queramos también es un punto a favor, ya que no nos limitamos con un servicio determinado.

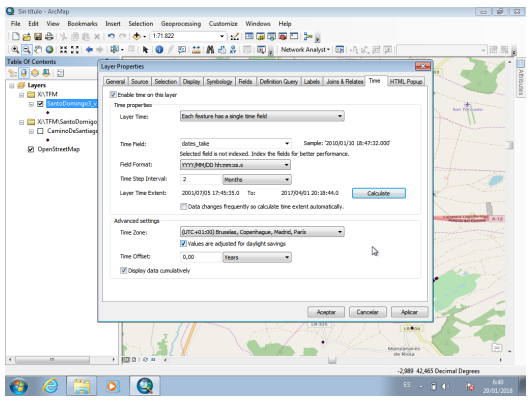
También está preparada para funcionar en smartphones, reconociendo los gestos típicos como pan o pinch sin tener que programar nada adicional.

2.5.- Uso de ArcGIS

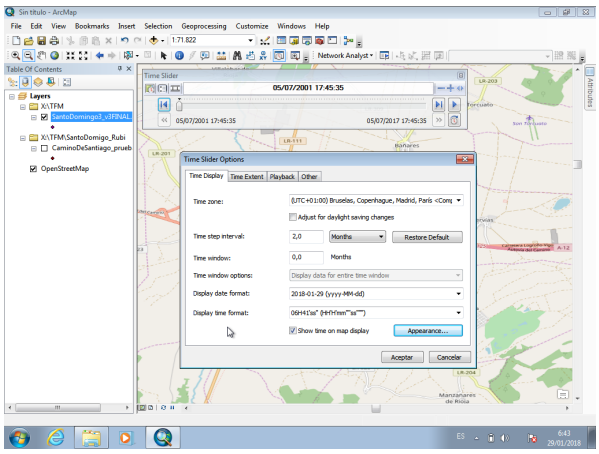
A través de ArcGis obtendremos un mapa de evolución de la toma de las fotografías a lo largo del tiempo haciendo uso de la configuración de un time slider.



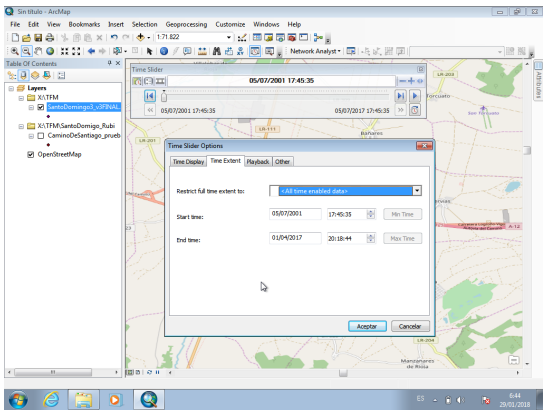
.- Configuraciones.



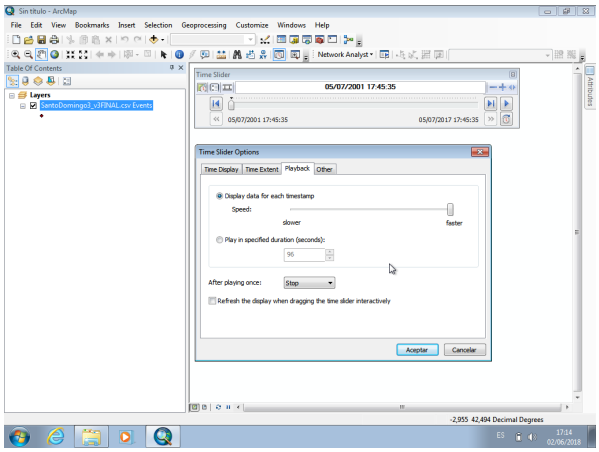
.- Configuraciones.



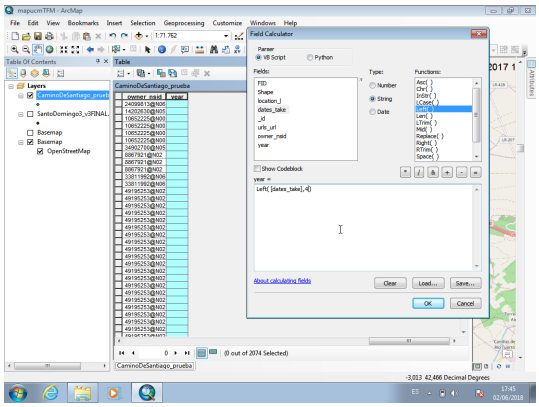
.- Configuraciones.



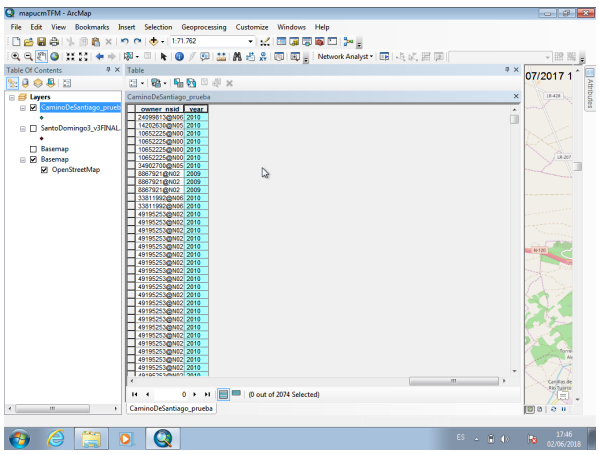
.- Configuraciones.



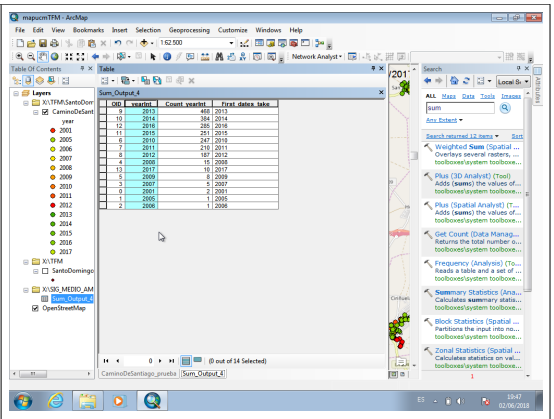
.- Configuraciones.



.- Cálculo de la columna year a partir del campo takes_date.



.- Resultado del campo year relleno.



.- Resultados con herramienta summarize por el campo year (count).

Mostraremos los resultados en nuestro mapa en la opción “videos”

https://s3.us-east-2.amazonaws.com/ucmap/videos.html

3.- Datos y tratamientos previos

3.1.- Datos iniciales e incorporación de coordenadas geográfica

Este paso en principio no debería ser necesario ya que partimos de datos geolocalizados, aun así realizamos la comprobación de la localización para eliminar aquellas fotos que ya no existan dentro de la red social Flickr. Pueden haber sido eliminadas por sus propietarios. De esta forma nos quedamos con datos todos válidos.

Partíamos de un shape file inicial con la información estructurada de la siguiente forma:

Table

CaminoDeSantiago_prueba

FID	Shape *	location_l	dates_take	id	urls_url	owner_nsid
0	Point	Santo Domingo de la Calzada	2010/01/10 18:47:32.000	4262480695	[{"type": "photopage", "_content":	24099813@N06
1	Point	Santo Domingo de la Calzada	2010/02/05 02:44:00.000	4331633889	[{"type": "photopage", "_content":	14202630@N05
2	Point	Santo Domingo de la Calzada	2010/03/20 15:40:48.000	4448050238	[{"type": "photopage", "_content":	10652225@N00
3	Point	Santo Domingo de la Calzada	2010/03/20 15:46:23.000	4448064550	[{"type": "photopage", "_content":	10652225@N00
4	Point	Santo Domingo de la Calzada	2010/03/20 15:50:48.000	4447299257	[{"type": "photopage", "_content":	10652225@N00
5	Point	Santo Domingo de la Calzada	2010/03/20 15:55:02.000	4447307459	[{"type": "photopage", "_content":	10652225@N00
6	Point	Santo Domingo de la Calzada	2010/04/02 11:46:19.000	4488307840	[{"type": "photopage", "_content":	34902700@N05
7	Point	Santo Domingo de la Calzada	2009/05/15 11:55:22.000	4542740759	[{"type": "photopage", "_content":	8867921@N02
8	Point	Santo Domingo de la Calzada	2009/05/15 15:15:21.000	4543374468	[{"type": "photopage", "_content":	8867921@N02
9	Point	Santo Domingo de la Calzada	2009/05/15 15:49:34.000	4543373496	[{"type": "photopage", "_content":	8867921@N02
10	Point	Santo Domingo de la Calzada	2010/04/26 21:02:53.000	4555616416	[{"type": "photopage", "_content":	33811992@N06
11	Point	Santo Domingo de la Calzada	2010/04/18 14:17:16.000	4575991718	[{"type": "photopage", "_content":	33811992@N06
12	Point	Santo Domingo de la Calzada	2010/05/31 00:00:06.000	4666145696	[{"type": "photopage", "_content":	49195253@N02
13	Point	Santo Domingo de la Calzada	2010/05/31 00:00:04.000	4666146384	[{"type": "photopage", "_content":	49195253@N02
14	Point	Santo Domingo de la Calzada	2010/05/31 00:00:02.000	4666146906	[{"type": "photopage", "_content":	49195253@N02
15	Point	Santo Domingo de la Calzada	2010/05/31 00:00:03.000	4665522359	[{"type": "photopage", "_content":	49195253@N02
16	Point	Santo Domingo de la Calzada	2010/05/31 00:00:07.000	4665521061	[{"type": "photopage", "_content":	49195253@N02
17	Point	Santo Domingo de la Calzada	2010/05/31 00:00:00.000	4665523215	[{"type": "photopage", "_content":	49195253@N02

.- Datos iniciales visualizados en ArcGis.

Para el propósito del trabajo los primeros pasos eran analizar la validez de los 2072 registros de las fotos de flickr, obtener y completar la información consiguiendo obtener una clasificación ideal para cada una de ellas a través de la herramienta de Google vision.

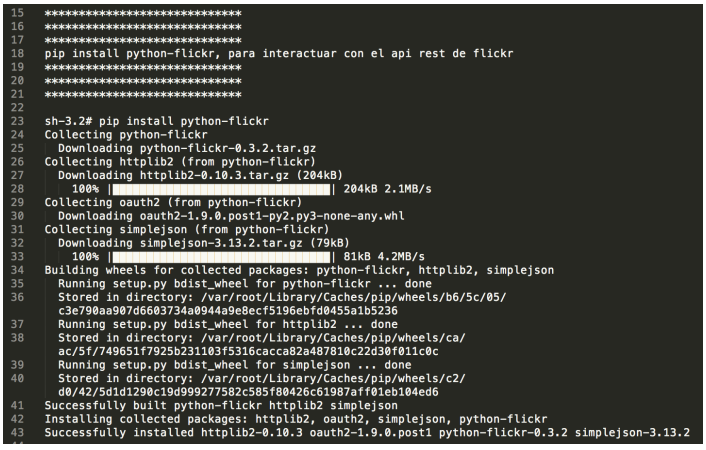
El primer paso fue exportar en formato CSV los datos del shape file antes nombrado para luego analizar mediante scripts de python todos sus datos.

Añadimos las columnas longitud y latitude a partir de los datos extraídos de flickr para cada una de las fotos.

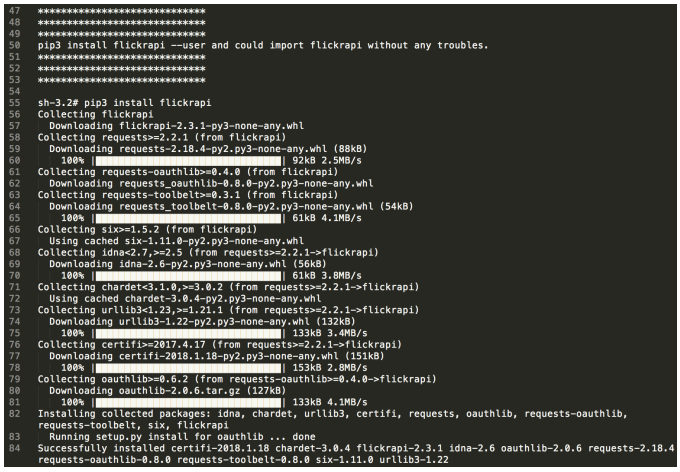
Hacemos referencia a la información aportada por Flickr en su web para el uso de la API para obtener la localización de cada foto:

https://www.flickr.com/services/api/explore/flickr.photos.geo.getLocation

Instalaciones iniciales en nuestra máquina para poder utilizar el api de flickr desde nuestros script de python:



.- Pip install python-flickr



.-Pip3 install fickrapi

Ejecutamos en python un programa para añadir dos columnas a nuestro fichero csv, con la longitude y latitude de cada foto extraída de Flickr API. Un ejemplo para un solo id de fotografía sería el siguiente:

En el script ejecutado de forma automática para todo el fichero csv controlamos que la obtención de los datos de longitud y latitud existan correctamente en flickr. Hay imágenes por ejemplo que han sido eliminadas y ya no tenemos que tenerlas en cuenta.

```python
#!/usr/bin/python

import flickrapi
import json
from pprint import pprint

api_key = 'f62de8ac68739905531c8378bfd4086b'
secret_api_key = '520216bfefa4089f'
photo_id = '6093476225'

flickr = flickrapi.FlickrAPI(api_key, secret_api_key)

try:
    photo = flickr.photos.geo.getLocation(api_key=api_key, photo_id=photo_id, format='parsed-json')

    m = photo
    n = json.dumps(m)
    o = json.loads(n)
    print(o['photo']['location'])

except flickrapi.exceptions.FlickrError as ex:
    print("Error code: %s" % ex.code)
    print("Error msg: %s" % ex)

```

.- Extracto de código python para consultar la localización de cada foto en Flickr.

Ejecución final del script (parte de final de la ejecución):

TOTAL= 2075, TOTAL OK= 2045, TOTAL KO= 29

```
EJECUCION FINAL CORRECTA PARA OBTENER LA LOCATION Y URLS EN COLUMNAS NUEVAS EN EL CSV
**********************************************************************************************
**********************************************************************************************
**********************************************************************************************

URL-->https://www.flickr.com/photos/116222242@N02/33783849805/
longitude=-2.919417 latitude=42.423241
2064 de 2075 ------ ID-PHOTO-FLICKR:33783847895
URL-->https://www.flickr.com/photos/116222242@N02/33783847895/
longitude=-2.917614 latitude=42.422136
2065 de 2075 ------ ID-PHOTO-FLICKR:33783856625
URL-->https://www.flickr.com/photos/116222242@N02/33783856625/
longitude=-2.902787 latitude=42.415719
2066 de 2075 ------ ID-PHOTO-FLICKR:33372268533
URL-->https://www.flickr.com/photos/trescastro/33372268533/
longitude=-2.953715 latitude=42.440771
2067 de 2075 ------ ID-PHOTO-FLICKR:33382879593
URL-->https://www.flickr.com/photos/trescastro/33382879593/
longitude=-2.953715 latitude=42.440771
2068 de 2075 ------ ID-PHOTO-FLICKR:33398273274
URL-->https://www.flickr.com/photos/cinglesdeberti/33398273274/
longitude=-2.951293 latitude=42.440578
2069 de 2075 ------ ID-PHOTO-FLICKR:34183333845
URL-->https://www.flickr.com/photos/trescastro/34183333845/
longitude=-2.953715 latitude=42.440771
2070 de 2075 ------ ID-PHOTO-FLICKR:34026011412
URL-->https://www.flickr.com/photos/trescastro/34026011412/
longitude=-2.953715 latitude=42.440771
2071 de 2075 ------ ID-PHOTO-FLICKR:33340913604
URL-->https://www.flickr.com/photos/trescastro/33340913604/
longitude=-2.953715 latitude=42.440771
2072 de 2075 ------ ID-PHOTO-FLICKR:34026004792
URL-->https://www.flickr.com/photos/trescastro/34026004792/
longitude=-2.953715 latitude=42.440771
2073 de 2075 ------ ID-PHOTO-FLICKR:34052394451
URL-->https://www.flickr.com/photos/trescastro/34052394451/
longitude=-2.953715 latitude=42.440771
2074 de 2075 ------ ID-PHOTO-FLICKR:34341506055
URL-->https://www.flickr.com/photos/san_torcuato_antes_villaporquera/34341506055/
longitude=-2.889175 latitude=42.481419
TOTAL= 2075
TOTAL OK= 2045
TOTAL KO= 29
```

.- Trazas de ejecución finales con valores totales de ok y ko añadiendo urls y location.

3.2- Limpieza de datos

Hemos querido unificar en un solo script en python todas las funcionalidades necesarias para poder obtener un fichero csv más completo y con información totalmente válida que nos sirva de punto de partida en el proceso que más adelante automatizaremos haciendo uso de la infraestructura de Amazon y sus servicios AWS.

Este script es un script escrito en python que hace uso de las librerías proporcionadas tanto por Google, Flickr, y AWS Amazon. Todas ellas establecen unos plazos de uso gratuitos de sus servicios mediante el registro como usuarios.

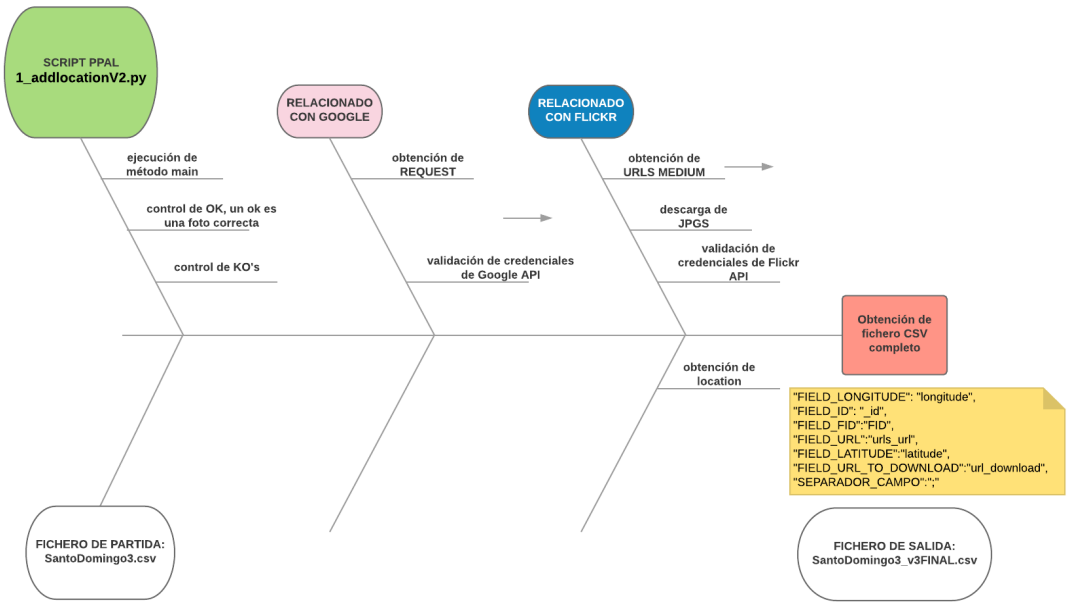
¿Qué hace nuestro script? paso a paso

Todo el código del proyecto lo hemos publicado en Git Hub y es de uso público:

https://github.com/rojorubi/ucm.sig.santiagomap

Código completo en github:
https://github.com/rojorubi/ucm.sig.santiagomap/blob/master/python/1_addlocationV2.py

1. Lee los datos de nuestro fichero de partida .csv (revisando errores de contenido, fotografías borradas en flickr e ignorándoles, contadores de ok y kos...).
2. Obtiene los datos de longitude y latitude de flickr junto con el valor de su url con formato medium.
3. Descarga la fotografía en nuestro local dentro del directorio photos.
4. Crea un fichero .json por cada foto con los datos de la request que necesita google cloud para obtener la clasificación automática de cada una de nuestras fotografías.
5. Completa nuestro fichero csv con los datos nuevos obtenidos.



.- Proceso completo para la obtención de un fichero csv con datos finales.

3.3. - Incorporación de las URLs y photos

Para poder pedir a través del API a Google la clasificación automática de nuestras fotografías, necesitamos tener almacenada una url válida en el proceso. Aprovechamos el mismo proceso para descargar en nuestra máquina las fotografías, por si en algún paso fuesen necesarias, aunque en principio este paso no sería necesario.

Las normas que establece flickr para sus urls son las siguientes:

Puedes crear la URL de origen a una foto una vez que sepas su ID., el ID. del servidor, el ID. del conjunto de servidores y el secreto, según lo indicado por muchos métodos API.

La URL toma el siguiente formato:

```
https://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}.jpg
      or
https://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}_[mstzb].jpg
      or
https://farm{farm-id}.staticflickr.com/{server-id}/{id}_{o-secret}_o.(jpg|gif|png)
```

Citando la documentación de la web de Flickr: Antes del 18 de noviembre de 2011, la API devolvía URL de imágenes con nombres de host como: "farm{farm-id}.static.flickr.com". Esas URL se siguen admitiendo.

Sufijos de tamaño:

Los sufijos de letras son los siguientes:

s, cuadrado pequeño 75x75

q, large square 150x150

t, imagen en miniatura, 100 en el lado más largo

m, pequeño, 240 en el lado más largo

n, small, 320 on longest side

-, mediano, 500 en el lado más largo

z, mediano 640, 640 en el lado más largo

c, tamaño mediano 800, 800 en el lado más largo

b, grande, 1024 en el lado más largo

h, grande de 1600, 1600 del lado más largo

k, grande de 2048, 2048 del lado más largo

o, imagen original, ya sea jpg, gif o png, según el formato de origen

La obtención de la url de cada una de nuestras fotografías la realizamos a partir de un script en python como explicamos a continuación:

```
#obtengo la url de la foto de flickr en calidad media para luego poder pasar en el curl a google y clasificar la imagen
def getUrlValidMediumSize(photo_id, size):
    '''
    EJEMPLO:
    https://farm1.staticflickr.com/2/1418878_1e92283336_m.jpg

    farm-id: 1
    server-id: 2
    photo-id: 1418878
    secret: 1e92283336
    size: m
    https://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}.jpg
    '''
    flickr = flickrapi.FlickrAPI(api_key, secret_api_key)
    try:
        photo = flickr.photos.getSizes(api_key=api_key, photo_id=photo_id, format='parsed-json')
        m = photo
        n = json.dumps(m)
        o = json.loads(n)
        #print(o['sizes'])
        sizes = o['sizes']['size']
        for s in sizes:
            label = s['label']
            if(label==size):
                url =s['source']
                if(url!="" or url!=null):
                    #print("URL VALID-->"+url)
                    return str(url)

    except flickrapi.exceptions.FlickrError as ex:
        print("Error code: %s" % ex.code)
        print("Error msg: %s" % ex)
        return "ko"
```

.- Extracto de código python para la obtención de los diferentes tamaños por foto en Flickr.

El código integrado en nuestro script general sería:El valor que obtenemos lo almacenamos para cada fotografía en el campo “url_download” en nuestro csv de datos que será cargado en dynamodb.

En paralelo podemos descargar las fotografías a partir del fichero siguiente que hemos montado en nuestro script:

```
ID;URL_VALIDMedium
4262480695;https://farm5.staticflickr.com/4029/4262480695_c43c734c22.jpg
4331633889;https://farm3.staticflickr.com/2752/4331633889_33ee729e7f.jpg
4448050238;https://farm5.staticflickr.com/4007/4448050238_8b19f31468.jpg
4448064550;https://farm5.staticflickr.com/4018/4448064550_515a10bd7d.jpg
4447299257;https://farm5.staticflickr.com/4005/4447299257_688af11c6d.jpg
4447307459;https://farm3.staticflickr.com/2772/4447307459_2e617e0465.jpg
4488307840;https://farm5.staticflickr.com/4013/4488307840_d978374f7e.jpg
4542740759;https://farm3.staticflickr.com/2436/4542740759_690a0b6bfc.jpg
4543374468;https://farm5.staticflickr.com/4070/4543374468_8c67354ecf.jpg
```

Por cada línea del fichero ejecutamos el siguiente código para guardar en local nuestras fotografías, en el subdirectorio photos/:

```
import requests
import string

my_consts={"FIELD_URL": "URL_VALIDMedium",
"FIELD_ID": "ID",
"SEPARADOR_CAMPO":";"}

DIRECTORY="photos/"

def downloadPhoto(url_photo, name_photo):
    img_data = requests.get(url_photo).content
    with open(name_photo, 'wb') as handler:
        handler.write(img_data)

# funcion para comprobar si existe o no el fichero introduccido por el usuario
def existe(nombreArch):
    try:
        f = open(nombreArch)
        f.close()
        return True
    except:
        return False

def getPhotos(fichero):
    #crear nuevo campo al final del fichero csv e ir rellenando el valor
    ficheroSalida = open(fichero, "r")
    lineas = ficheroSalida.readlines()
    ficheroSalida.close()

    fichero_sinextension = fichero.split(".")

    lineas[0] = lineas[0].rstrip()
    listaCabecera = lineas[0].split(my_consts["SEPARADOR_CAMPO"]) # tengo separadas todas las etiquetas de la cabecera
    indice = listaCabecera.index(my_consts["FIELD_ID"])
    indicePhoto = listaCabecera.index(my_consts["FIELD_URL"])
    contadorOK=0
    contadorKO=0

    contador = 0
    print("total lineas")
    print(len(lineas))
    for cont in range(len(lineas)):
        lineas[cont] = lineas[cont].rstrip()
        listaValores = lineas[cont].split(my_consts["SEPARADOR_CAMPO"])
        idPhoto = listaValores[indice]
        urlPhoto = listaValores[indicePhoto]
        if(idPhoto!=""):
            print(str(cont)+" de "+str(len(lineas))+" - ID-PHOTO-FLICKR:"+idPhoto+" - URL-PHOTO-FLICKR:"+urlPhoto)
            if cont!=0:
                downloadPhoto(urlPhoto, idPhoto+".jpg")

    print("TOTAL= "+str(len(lineas)))
    print("TOTAL OK= "+str(contadorOK))
    print("TOTAL KO= "+str(contadorKO))

    ficheroSalida.close()


def main(nombreFichero):
    nombreF = nombreFichero
    while not existe(nombreF):
        # pedir nombres de ficheros hasta que el usuario introduzca uno que exista
        gp.AddMessage("el fichero no existe")
    getPhotos(nombreF)


main("listIdPhotosAndUrlsValid.txt")
```

.- Function getPhotos para leer la url de cada foto y descargarla en local con la función downloadPhoto.

4.- Google Cloud Vision para clasificación automática de imágenes

A través de las herramientas que nos proporciona Google vision vamos a establecer un proceso para obtener de todas nuestras imágenes válidas una clasificación automática que consistirá en una serie de tags valoradas con una puntuación de aproximación a los objetos y características de nuestras fotografías, esta valoración google la llama "score".

Para lleva a cabo la clasificación utilizaremos Google vision que ejecutaremos a partir de los registros que tenemos guardados en nuestra base de datos no relacional en Amazon (DynamoDB) por cada uno de los registros y al hacer un put (o guardar en S3 un fichero con nuestra petición/request) de la request necesaria para hacer la petición a Google de nuestra clasificación. Este put lanzará o ejecutará una lambda (proceso en python) que obtendrá la response de Google y que será almacenada en un nuevo campo "clasification_google_vision" por cada uno de nuestros registros (correspondiente con los datos de una foto) con todos los valores de reconocimiento automático que hemos obtenido a través de Google.

Las características principales de Google Vision son las siguientes: En nuestro trabajo utilizaremos **Detección de etiquetas y Detección web.**

Extrae información valiosa a partir de las imágenes con nuestra potente API Vision de Cloud**.**

Hacemos referencia al listado de características que nos proporciona Google Vision en su web:

https://cloud.google.com/vision/

Detección de etiquetas. Detecta amplios conjuntos de categorías dentro de una imagen, desde medios de transporte hasta animales.

Detección de contenido explícit. Detecta contenido explícito, por ejemplo, contenido para adultos o contenido violento.

Detección de logotipos. Detecta logotipos de productos muy conocidos dentro de una imagen.

Detección de puntos de referencia. Detecta estructuras artificiales y naturales muy famosas dentro de una imagen.

Reconocimiento óptico de caracteres. Detecta y extrae texto de una imagen. Esta función es compatible con un gran número de idiomas y con la identificación automática de idiomas.

Detección de caras. Detecta varias caras en una imagen, además de sus atributos faciales clave, como el estado emocional o las prendas que lleva en la cabeza.**No se ofrece reconocimiento facial.**

Atributos de la imagen. Detecta los atributos generales de la imagen, como el color dominante, y ofrece sugerencias de recorte pertinentes.

Detección web. Busca imágenes similares en Internet.

API REST integrada. Accede mediante la API REST para solicitar uno o varios tipos de anotación por imagen. Las imágenes pueden subirse en la solicitud o integrarse en Google Cloud Storage.

Obtención de todos los ficheros json con la request que necesitamos preguntar a Google. Nuestra ejecución será a través de una llamada curl a Google:

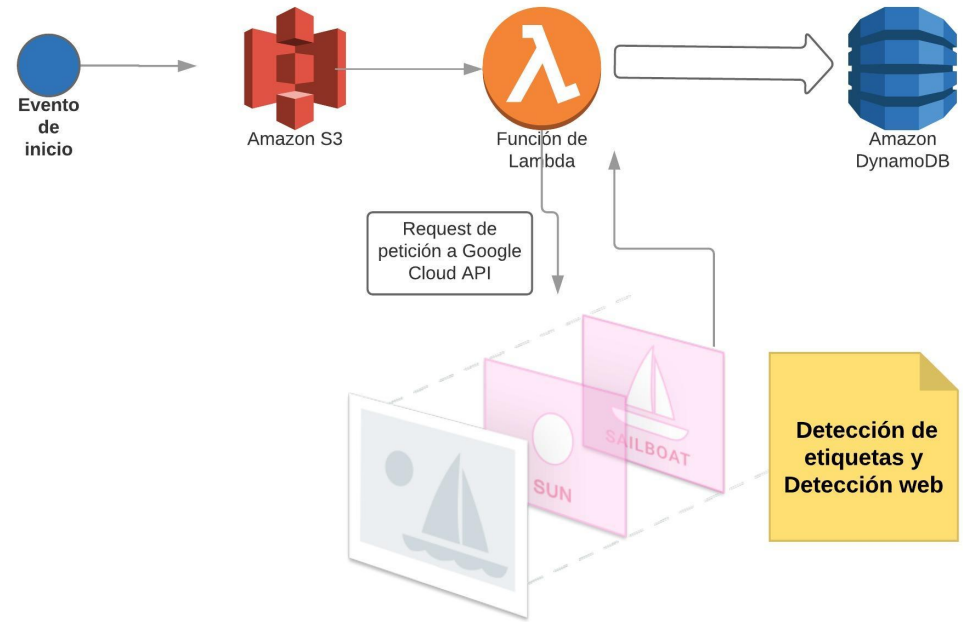
A partir de nuestra API_KEY ejecutaremos el curl de la siguiente manera:

curl -v -s -H "Content-Type: application/json" \
https://vision.googleapis.com/v1/images:annotate?key=API_KEY \
--data-binary @request.json

Esto lo traduciremos en nuestro codigo python de nuestra lambda en aws.

En nuestro script el código integrado es el siguiente:

```
def createJsonFileToSendS3(idPhoto, urlPhoto):
    ficheroSalida = open("filesJsonFinalesCurl/"+idPhoto+".json", "w")

    linea = '{"requests":[{"image":{"source": {"imageUri":"'+urlPhoto+'"}},"features":[{"type":"LANDMARK_DETECTION","maxResults":1},{"type":"WEB_DETECTION","maxResults":3}]}]}';

    print("linea: "+linea)

    linea_json=json.dumps(json.loads(linea))

    #linea = 7{'id':'"+idPhoto+"','url':'"+urlPhoto+"'}"
    ficheroSalida.write(linea_json)
    ficheroSalida.close()
```



En Amazon S3 guardamos nuestra request que utilizaremos a la hora de llamar a Google, ejemplo de petición, que será un fichero json con las siguientes características:

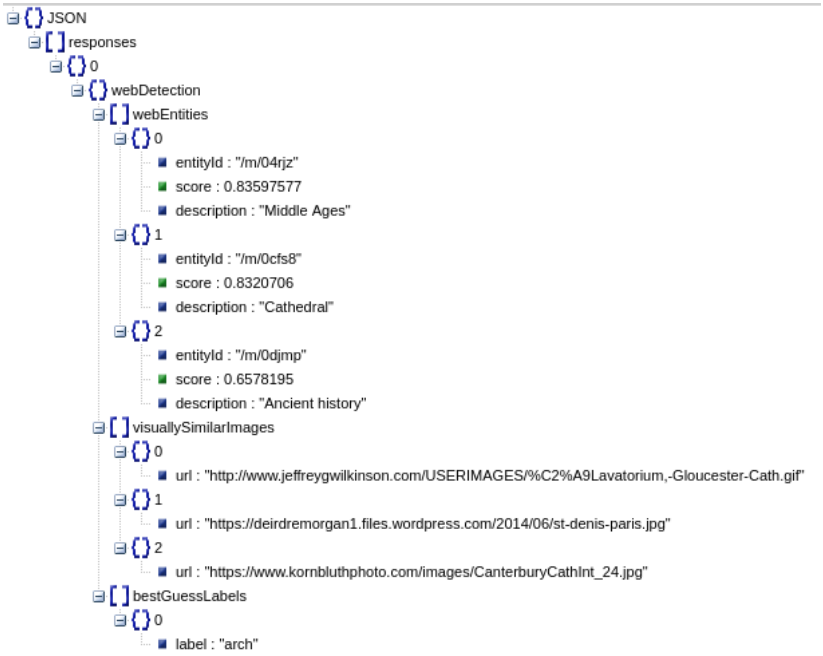
https://s3.us-east-2.amazonaws.com/requestgooglecloud2/10153021214.json (el nombre del fichero es el id de la foto en flickr)

Landmark Detection detecta las estructuras naturales y artificiales populares dentro de una imagen. (https://cloud.google.com/vision/docs/detecting-landmarks)
Detecting Web Entities and Pages. Web Detection detecta referencias web a una imagen. (https://cloud.google.com/vision/docs/detecting-web)

Nuestra **función lambda** ejecutará el siguiente script de python:

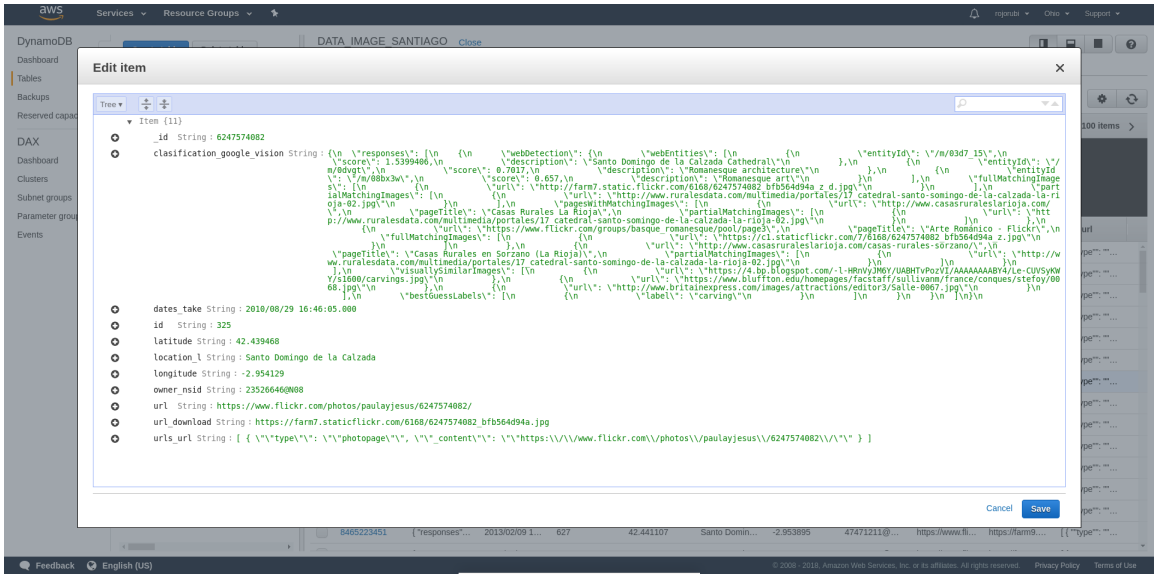
```python
import json
import urllib2
import os
import boto3
from boto3.dynamodb.conditions import Key, Attr

def updateRowInDynamoDB(idphoto, clasification):
    table = boto3.resource('dynamodb').Table('DATA_IMAGE_SANTIAGO')
    response = table.update_item(
        Key={
            '_id': str(idphoto)
        },
        UpdateExpression="set clasification_google_vision = :c",
        ExpressionAttributeValues={
            ':c': str(clasification)
        },
        ReturnValues="UPDATED_NEW"
    )
    print "UpdateItem succeeded:"
    print(json.dumps(response, indent=4))
    print "end process."

def callingToGoogle(idphoto, url_process_lead, payload_json):
    print "Calling to google..."
    req = urllib2.Request(url_process_lead, payload_json)
    req.add_header("Content-type", "application/json")
    res = urllib2.urlopen(req)
    response = res.read()
    print "Response to GOOGLE CODE CLASIFICATION IMAGE:",response
    #buscar en la tabla de dynamo el  id de la foto con idphoto e insertar el valor del campo clasification_google_vision
    updateRowInDynamoDB(idphoto, response)

#https://github.com/gxx/aws-lambda-python/blob/master/ARTICLE.md
def lambda_handler(event, context):
    print "Event:",event
    # Obtain the bucket name and key for the event
    bucket_name = event['Records'][0]['s3']['bucket']['name']
    print "bucket_name:",bucket_name
    key_path = event['Records'][0]['s3']['object']['key']
    print "key_path:",key_path
    idphoto = key_path.split(".")[0]
    print "idphoto:",idphoto
    payload_json_s3 = boto3.resource('s3').Object(bucket_name, key_path).get()['Body'].read() # Retrieve the S3 Object
    print "body from s3 json:",payload_json_s3
    url_process_lead = 'https://vision.googleapis.com/v1/images:annotate?key=AIzaSyBAkCyuOZpyerrB-d2Fo4dm2Egw8dFm_SM'
    payload_json=json.dumps(json.loads(payload_json_s3))
    callingToGoogle(idphoto, url_process_lead, payload_json)
```

Obtendremos un resultado de ejemplo como el siguiente:



Que almacenaremos en DynamoDB:
Campos y descripción de cada valor que almacenamos:

{
"_id": "6247574082", //ID DE LA FOTOGRAFÍA EN FLICKR
"clasification_google_vision": "**********", //RESULTADO DE CLASIFICACIÓN OBTENDIO CON GOOGLE CLOUD
"dates_take": "2010/08/29 16:46:05.000", //FECHA EN LA QUE HA SIDO HECHA LA FOTO
"id": "325", // ID UNICO QUE IDENTIFICA EL REGISTRO EN DYNAMODB
"latitude": "42.439468", // DATOS DE LOCALIZACIÓN
"location_l": "Santo Domingo de la Calzada", // LUGAR DONDE FLICKR TIENE LOCALIZADA LA FOTO
"longitude": "-2.954129", // DATOS DE LOCALIZACIÓN
"owner_nsid": "23526646@N08", // ID DEL PROPIETARIO DE LA FOTO EN FLICKR
"url": "https://www.flickr.com/photos/paulayjesus/6247574082/", // URL EN FLICKR
"url_download": "https://farm7.staticflickr.com/6168/6247574082_bfb564d94a.jpg", // URL DE LA FOTO INDEPENDIENTE QUE PUEDE SERVIRNOS PARA DESCARGAR LA FOTO
"urls_url": "[{ \"\"type\"\": \"\"photopage\"\", \"\"_content\"\": \"\"https:\\/\\/www.flickr.com\\/photos\\/paulayjesus\\/6247574082\\/\"\" }]" // VALORES DE URLS ORIGINAL}



Una vez obtenida la información de clasificación automática tendríamos toda la información completa por cada foto para poder analizar de forma conjunta la clasificación de todas nuestras fotografías.
Todos los registros guardados en nuestro DynamoDB serán los datos que consultaremos desde nuestra página web para mostrar de forma individual en nuestro mapa y también de forma conjunta.
A partir de estos datos analizaremos los conjuntos de tags o clasificaciones más recurrentes entre nuestros datos, las menos recurrentes y extraeremos los ficheros con extensión GEOJSON que nos servirán para poder exportarlos a otros programas de análisis como ArcGis.

Ejemplo con imagen: a partir de la siguiente imagen y la request que enviamos a google recibimos una clasificación específica.

Request a google:
https://s3.us-east-2.amazonaws.com/requestgooglecloud2/6247574082.json

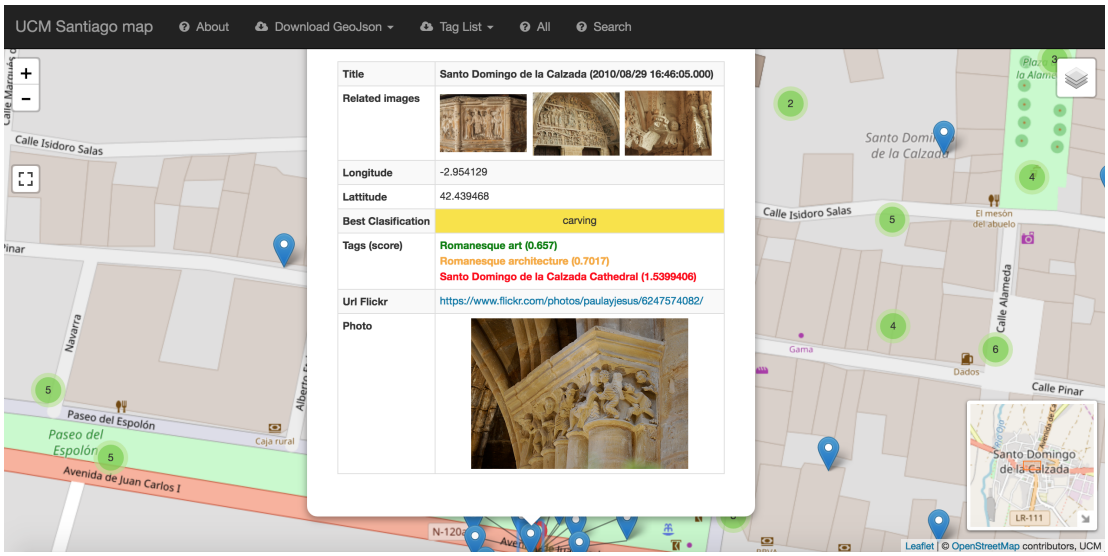
```
{
	"requests": [{
		"image": {
			"source": {
				"imageUri":
"https://farm7.staticflickr.com/6168/6247574082_bfb564d94a.jpg"
			}
		},
		"features": [{
			"type": "LANDMARK_DETECTION",
			"maxResults": 1
		}, {
			"type": "WEB_DETECTION",
			"maxResults": 3
		}]
	}]
}
```


Response con clasificación:


```
{
"responses": [{
	"webDetection": {
		"webEntities": [{
			"entityId": "/m/03d7_15",
			"score": 1.5399406,
			"description": "Santo Domingo de la Calzada Cathedral"
		}, {
			"entityId": "/m/0dvgt",
			"score": 0.7017,
			"description": "Romanesque architecture"
		}, {
			"entityId": "/m/08bx3w",
			"score": 0.657,
			"description": "Romanesque art"
		}],
		"fullMatchingImages": [{
			"url":
"http://farm7.static.flickr.com/6168/6247574082_bfb564d94a_z_d.jpg"
		}],
		"partialMatchingImages": [{
			"url":"http://www.ruralesdata.com/multimedia/portales/17_cate
dral-santo-somingo-de-la-calzada-la-rioja-02.jpg"
		}],
		"pagesWithMatchingImages": [{
			"url": "http://www.casasruraleslarioja.com/",
			"pageTitle": "Casas Rurales La Rioja",
			"partialMatchingImages": [{
				"url":
"http://www.ruralesdata.com/multimedia/portales/17_catedral-santo-somingo-de-la-
calzada-la-rioja-02.jpg"
			}]
		}, {
			"url":
"https://www.flickr.com/groups/basque_romanesque/pool/page3",
			"pageTitle": "Arte Románico - Flickr",
			"fullMatchingImages": [{
				"url":
"https://c1.staticflickr.com/7/6168/6247574082_bfb564d94a_z.jpg"
			}]
		}, {
			"url": "http://www.casasruraleslarioja.com/casas-rurales-
sorzano/",
			"pageTitle": "Casas Rurales en Sorzano (La Rioja)",
			"partialMatchingImages": [{
```

```
                        "url":
"http://www.ruralesdata.com/multimedia/portales/17_catedral-santo-somingo-de-la-
calzada-la-rioja-02.jpg"
                    }]
            }],
            "visuallySimilarImages": [{
                    "url": "https://4.bp.blogspot.com/-l-
HRnVyJM6Y/UABHTvPozVI/AAAAAAAABY4/Le-CUVSyKWY/s1600/carvings.jpg"
            }, {
                    "url":
"https://www.bluffton.edu/homepages/facstaff/sullivanm/france/conques/stefoy/0068
.jpg"
            }, {
                    "url":
"http://www.britainexpress.com/images/attractions/editor3/Salle-0067.jpg"
            }],
            "bestGuessLabels": [{
                    "label": "carving"
            }]
        }
}]
}
```


Ejemplo de nuestra imagen ya clasificada cargada en nuestro mapa resultado:



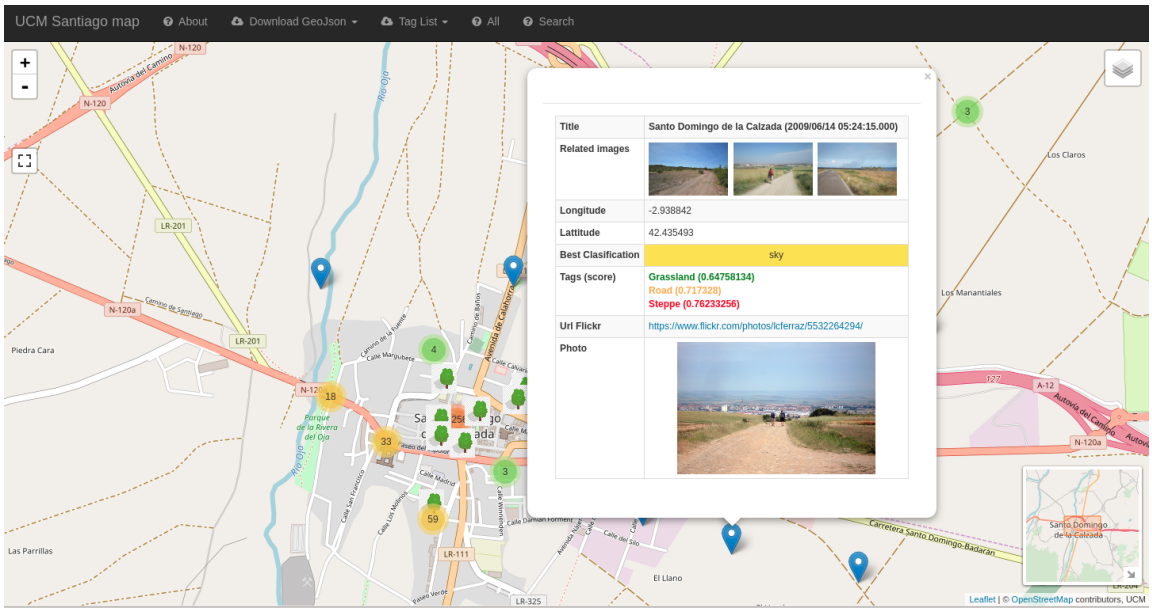
Otros ejemplos de datos cargados en nuestro mapa:

Ejemplo de clasificación sobre un paisaje exterior:

Request para obtener la clasificación con Google Vision:

https://s3.us-east-2.amazonaws.com/requestgooglecloud2/5532264294.json


```
{"requests": [{
              "image": {
                     "source": {
                            "imageUri":
"https://farm6.staticflickr.com/5058/5532264294_0492a06956.jpg"
                     }
              },
              "features": [{
                     "type": "LANDMARK_DETECTION",
                     "maxResults": 1
              }, {
                     "type": "WEB_DETECTION",
                     "maxResults": 3}]}]}
```

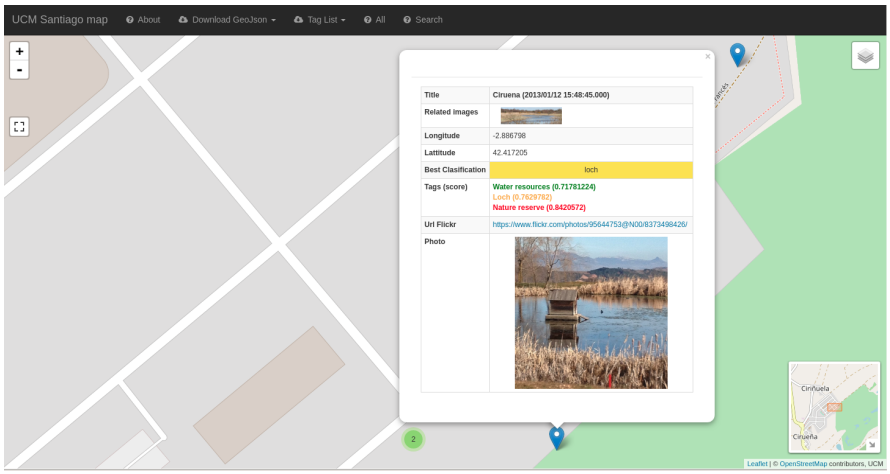
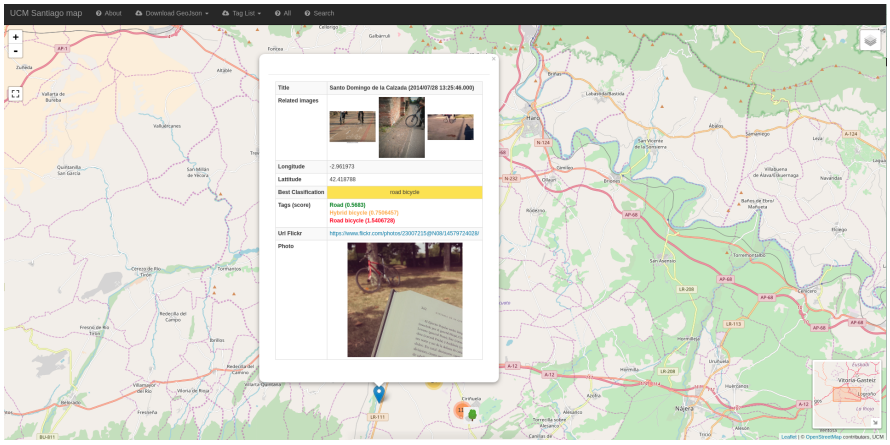



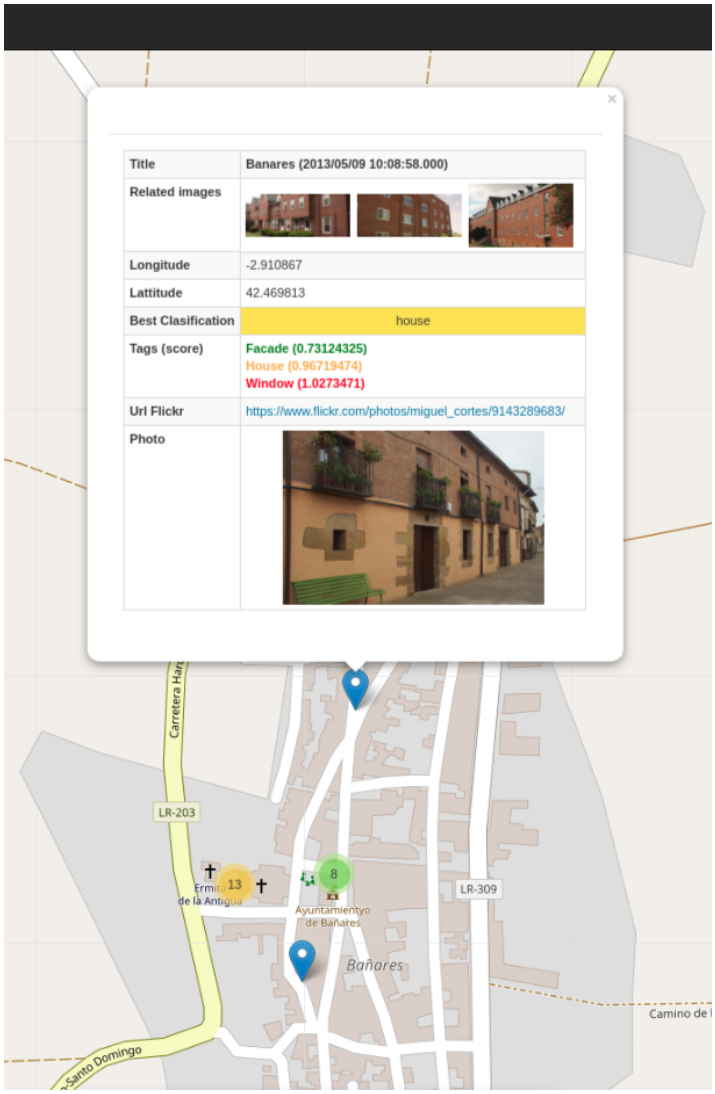
Reponse con clasificación:


```
{
       "responses": [{
              "webDetection": {
                     "webEntities": [{
                            "entityId": "/m/0f45l",
                            "score": 0.76233256,
                            "description": "Steppe"
                     }, {
                            "entityId": "/m/06gfj",
                            "score": 0.717328,
                            "description": "Road"
                     }, {
                            "entityId": "/m/01c7cq",
                            "score": 0.64758134,
                            "description": "Grassland"
                     }],
                     "visuallySimilarImages": [{
                            "url":
"https://www.outdoorproject.com/sites/default/files/styles/cboxshow/public/1493657454/5f7a3321_lava_flow_hawaii_volcanoes_national_park_halvor_tweto.jpg?itok=chXqD4ar"
                     }, {
                            "url":"http://www.trekkingtrutime.com/wp-content/uploads/2015/12/cam1.jpg"
                     }, {
```

```
                        "url":
"https://images.trailsoffroad.com/trail_descriptions/photo2s/000/025/711/large/2.
7.jpg?1500326305"
                    }],
                    "bestGuessLabels": [{
                        "label": "sky"
                    }]
                }
        }]
}
```


Otros ejemplos de clasificaciones en exteriores:

	"Nature reserve" (0.8420572) "Loch" (0.7629782) "Water resources" (0.71781224) "bestGuessLabels":"loch"
	“Road bicycle" (1.5406728) "Hybrid bicycle" (0.7506457) "Road" (0.5683) "bestGuessLabels": "road bicycle"



"Window" (1.0273471)

"House" (0.96719474)

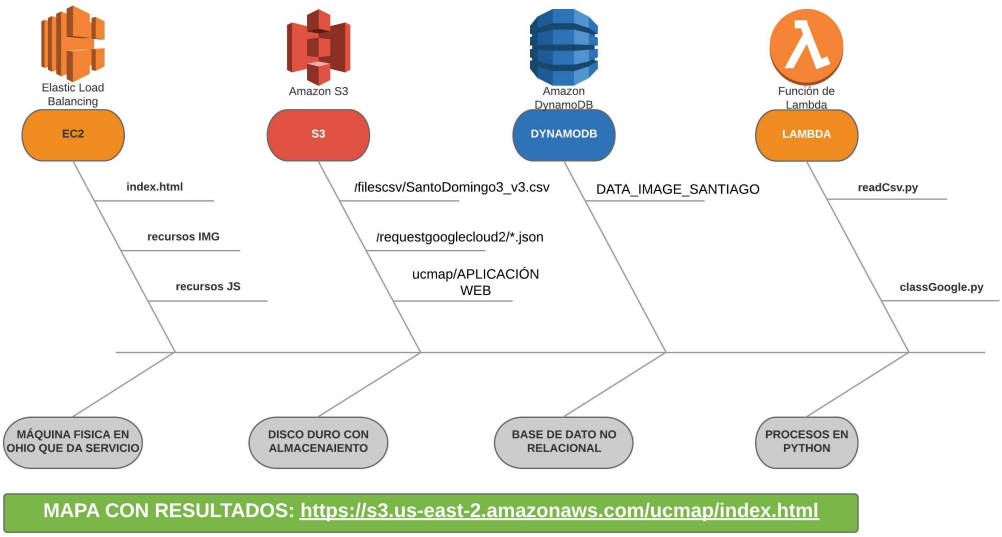
"Facade" (0.73124325)

"bestGuessLabels": "house"

5.- Procesos batch para el guardado de datos, AWS infraestructura

¿Por qué elegimos aws y que función tiene en nuestro proyecto?

AWS amazon nos proporciona la infraestructura necesaria para poder, tanto dar servicio web a nuestra página web, como poder ejecutar los procesos necesarios para llamar a google, como el soporte necesario de una base de datos no relacional que almacenará todos los datos que cargaremos a partir de nuestro fichero csv con toda la información completada (SantoDomingo3_v3FINAL.csv).



.- Esquema del uso de web services de Amazon en todo el proyectxo.

1. Carga inicial de nuestro fichero csv completo SantoDomingo3_v3FINAL.csv en S3 (https://s3.eu-west-3.amazonaws.com/filescsv/SantoDomingo3_v3.csv)
2. Ejecución de un proceso py dentro de una lambda que se ejecutará manualmente que leerá ese fichero csv y almacenará por cada una de sus lineas un registro en base de datos en dynamodb en la tabla DATA_IMAGE_SANTIAGO

```python
import boto3
import csv

'''
lambda para la carga inicial de datos del csv en la tabla de dynamodb
'''
def lambda_handler(event, context):
    s3 = boto3.resource('s3')
    dynamodb = boto3.resource('dynamodb')

    bucket = s3.Bucket('filescsv')
    object_key = 'SantoDomingo3_v3.csv'

    obj = bucket.Object(object_key)
    print('Bucket name: {}'.format(bucket.name))
    print('Object key: {}'.format(obj.key))
    print('Object content length: {}'.format(obj.content_length))
    print('Object body: {}'.format(obj.get()['Body'].read()))
    print('Object last modified: {}'.format(obj.last_modified))

    lines = obj.get()['Body'].read().split(b'\n')
    #for r in lines:
       #print(r.decode())

    table =dynamodb.Table('DATA_IMAGE_SANTIAGO')

    #FID;location_l;dates_take;_id;urls_url;owner_nsid;longitude;latitude;url;url_download
    with table.batch_writer() as batch:
        cont=1
        for row in lines:
            #print(row.decode().split(';')[1])
            batch.put_item(Item={
                'id':str(cont),
                'location_l':row.decode().split(';')[1],
                'dates_take':row.decode().split(';')[2],
                '_id':row.decode().split(';')[3],
                'urls_url':row.decode().split(';')[4],
                'owner_nsid':row.decode().split(';')[5],
                'longitude':row.decode().split(';')[6],
                'latitude':row.decode().split(';')[7],
                'url':row.decode().split(';')[8],
                'url_download':row.decode().split(';')[9]
            })
            cont+=1
```

3. Subimos de forma manual todas los ficheros idphoto.json con el valor de cada request por cada foto de la que queremos obtener su clasificación a partir de google cloud vision, los almacenamos en el directorio en S3 siguiente arn:aws:s3:::requestgooglecloud2

4. Ejecutamos otra lambda que se ejecutara de forma individual por cada fichero json que hemos subido al S3 y que obtendrá de Google la clasificación y completará la información en el registro de dynamodb correspondiente.

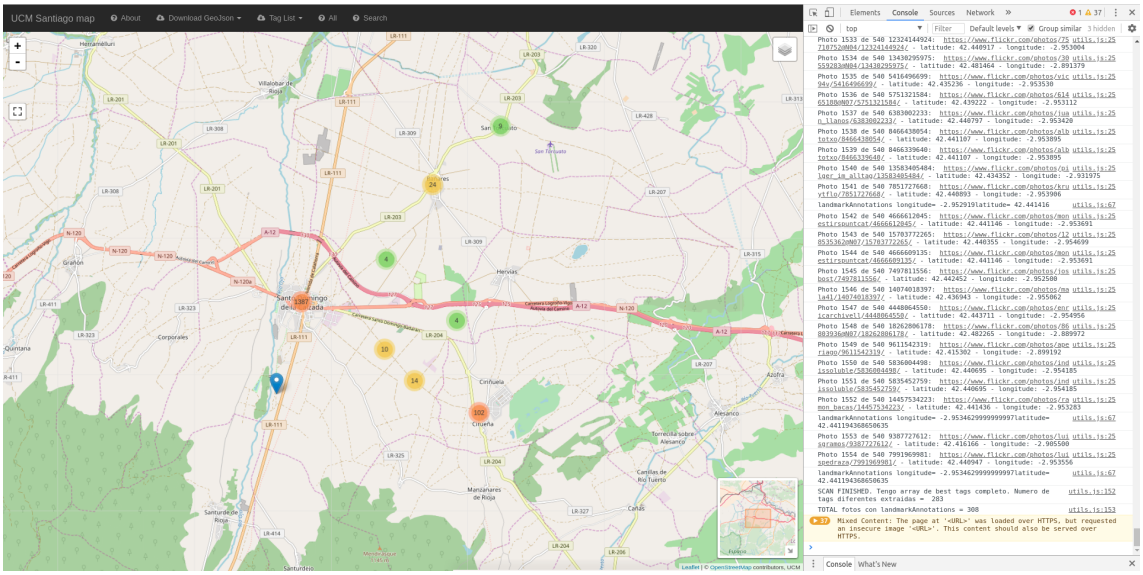
6.- Análisis y primeras visualizaciones de datos sobre un mapa base. Renderización en Java Script con leaflet.

Parte de visualización de datos para analizar sobre un mapa los datos extraídos.

https://s3.us-east-2.amazonaws.com/ucmap/index.html

Utilizaremos un mapa base basado en open street maps y análisis por capas con la librería leaflet (javascript). A partir de funciones javascript manejaremos los datos, un resumen de las interacciones que podemos hacer con nuestro mapa resultado son las siguientes:

1. Cargar de forma incremental los 1554 datos que tenemos guardados en nuestra base de datos no relacional en AWS dynamodb.

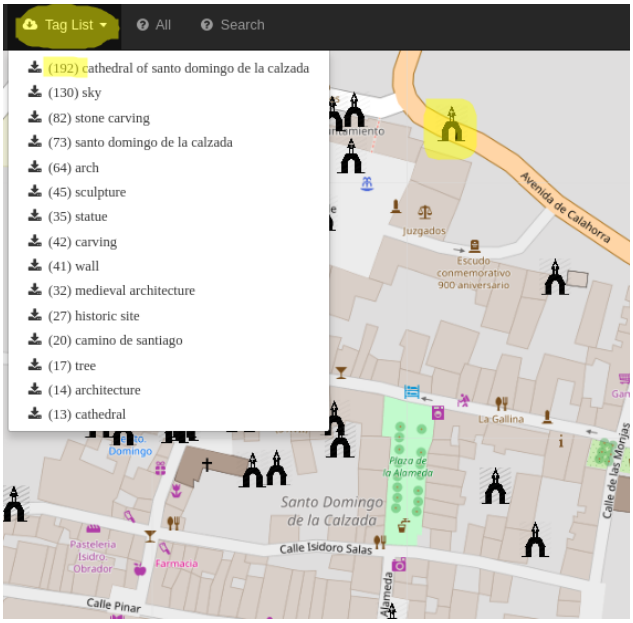


.- Ejemplo de carga con clustering y consola con trazas de carga con los datos procedentes de DynamoDB.

Mostramos en el mapa de inicio de forma incremental todo los datos que tenemos guardados en dynamodb. Ayudándonos de la consola del navegador web y la configuración de trazas que hemos añadido en nuestro código fuente pordemos ir visualizando los datos exactos que estan siendo añadidos de forma dinámica a nuestro mapa.

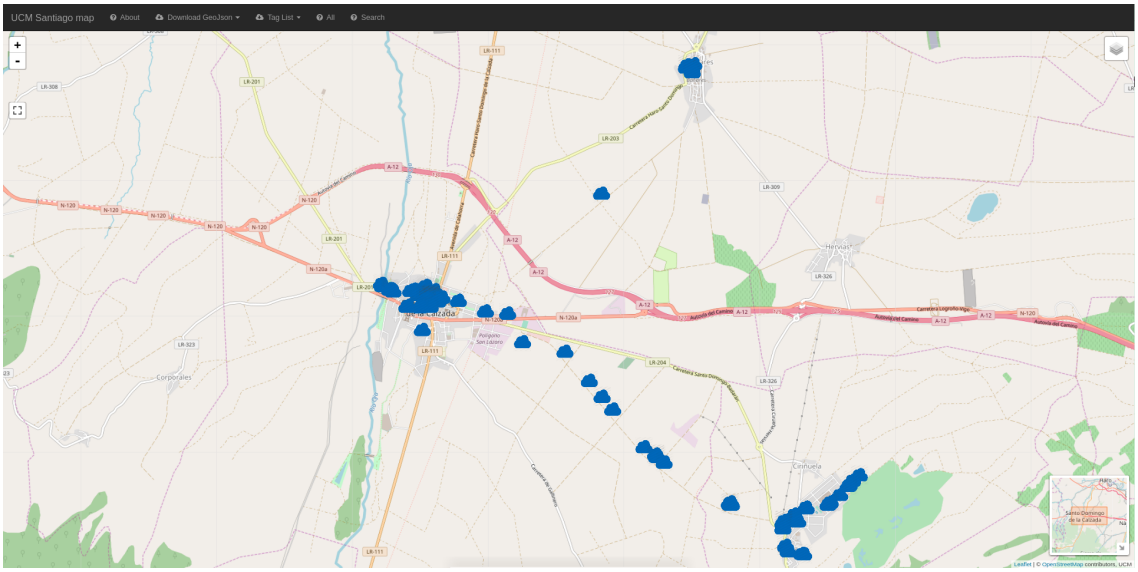
Nos servimos de la herramienta de lógica de múltiples capas de marcadores basado en leaflet, “leaflet marker cluster plugin”.

2. Cargar de forma individual las fotografías clasificadas en las primeras 15 tags más recurrentes entre nuestras fotos.

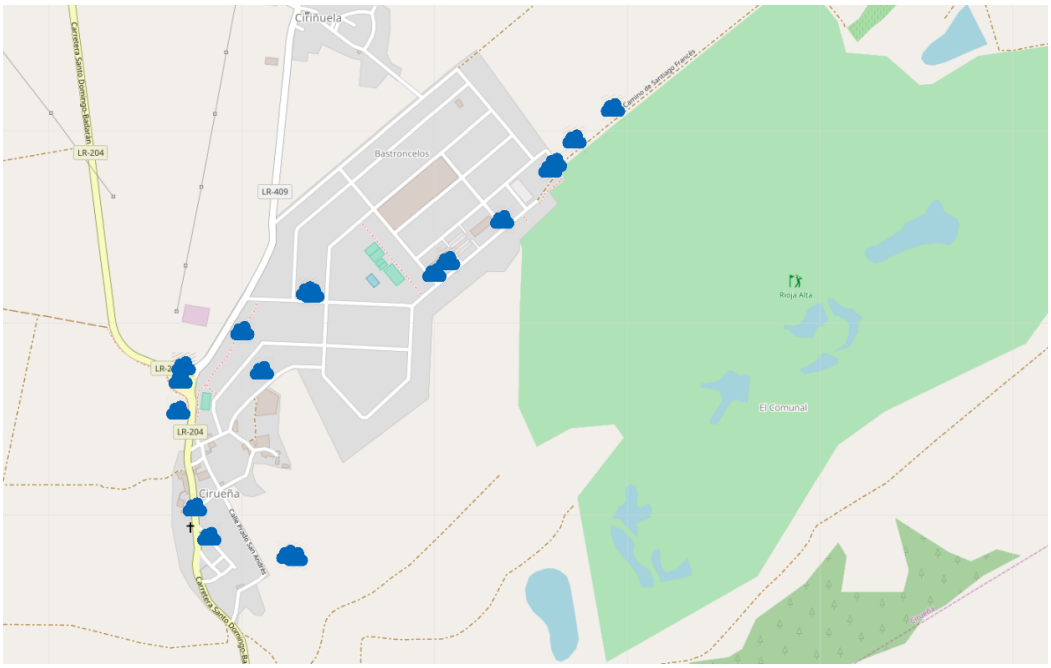


.- Combo selector de ficheros geoJson.

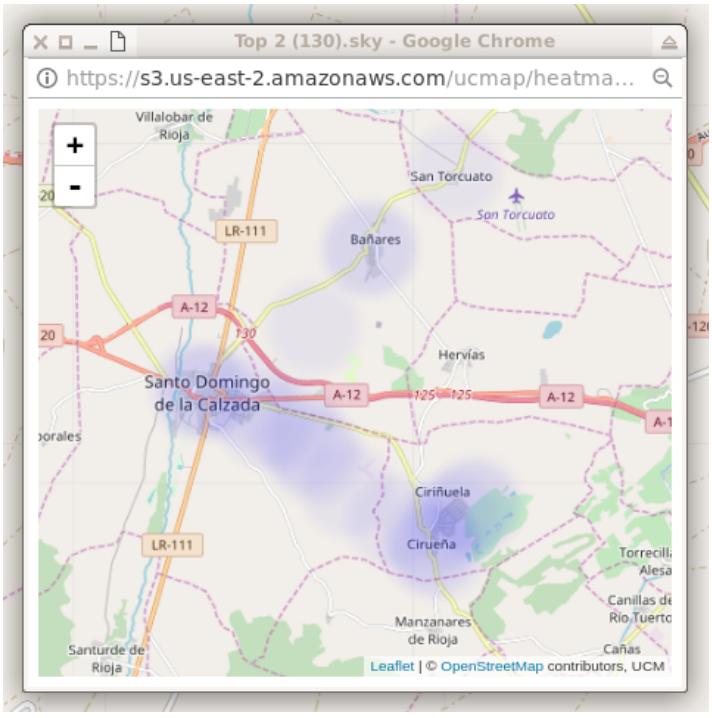
Ejemplo de carga para la tag “sky”:



.- Distribución de las fotografías clasificadas como “sky” con icono personalizado para estos puntos.

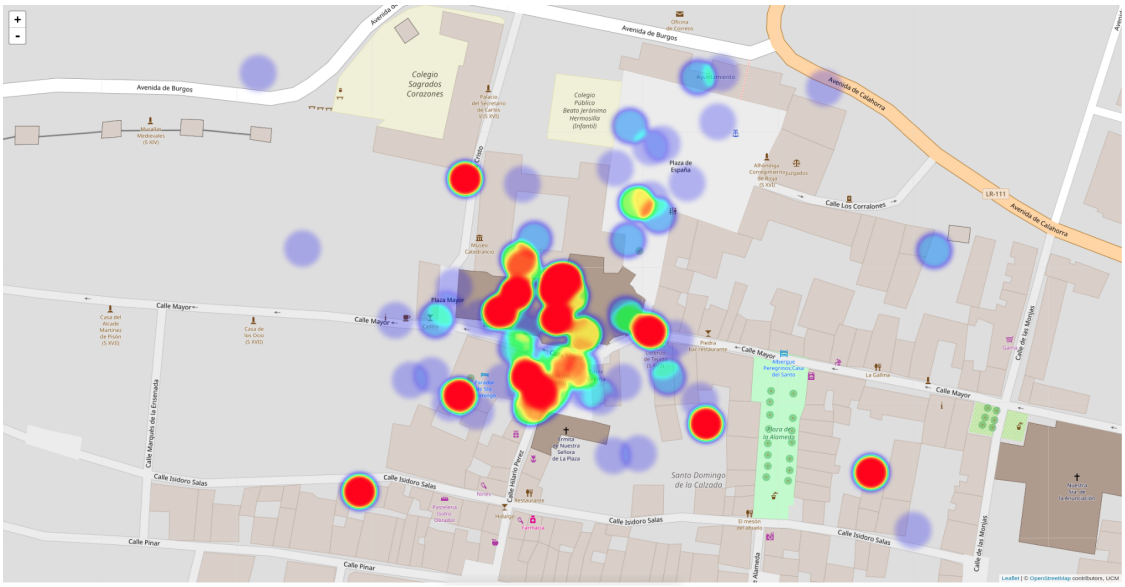


.- Detalle con zoom de las fotografías clasificadas como “sky” en la localidad de Cirueña.

	En el mapa de calor podemos ver la distribución más genérica de las fotografías clasificadas como “sky” como mejor clasificación obtenida. Vemos que las localidades de Santo Domingo de la Calzada y Cirueñela son las más fotografiadas. Por la distribución también podemos deducir que esas fotografías sobre el cielo han sido realizadas a través de los caminos que comunican las principales localidades.

3. Heat maps para las 15 primeras tags más repetidas.

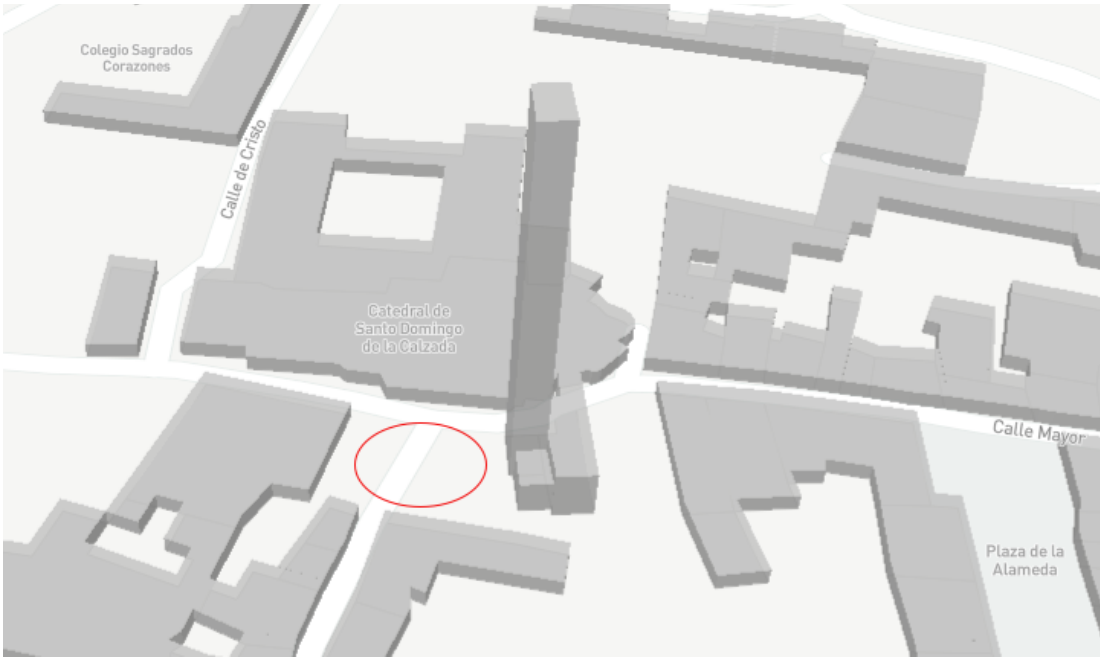
   El objetivo de tener mapas de calor es poder visualizar de forma más rápida la distribución de las fotografías. Tomamos como ejemplo las 192 fotografías que tenemos clasificadas en el primer lugar como “cathedral of santo domingo de la calzada”



.- Heat map para la tag en posición 1 “cathedral of santo domingo de la calzada”.

.- Detalle de la plaza del Santo.

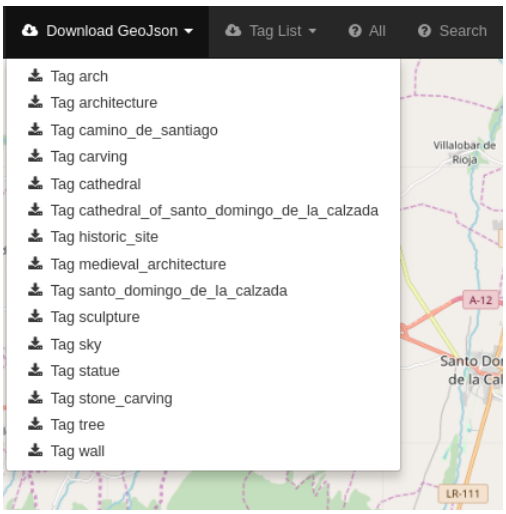
En el análisis del mapa de calor de la tag “cathedral of santo domingo de la calzada” podemos observar que los puntos más fotografiados se encuentran dentro del propio edificio y existe un segundo punto localizado fuera que coincide con la plaza que precede a la entrada a la catedral. Coincide con el comportamiento de un turista habitual.



.- Detalle de la plaza del Santo en nuestro mapa en 3D.

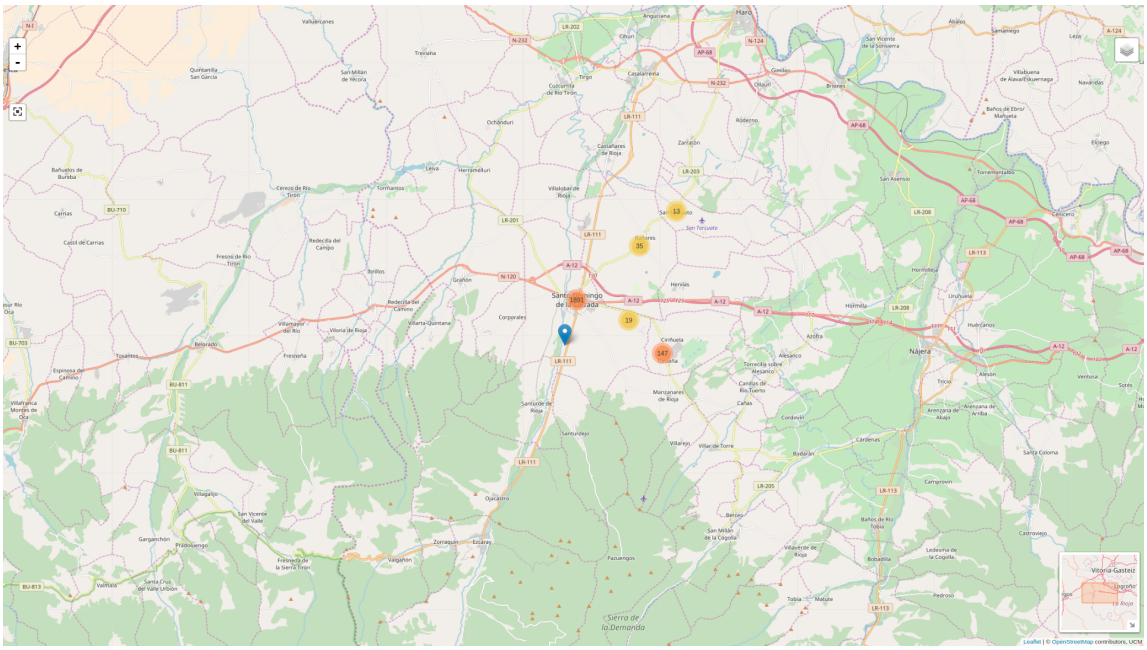
4. Descargar los geojson resultado a nuestro directorio local.

   Tenemos disponibles los geojson para poder exportarlos a otras herramientas de GIS como ArcGIS u otras.

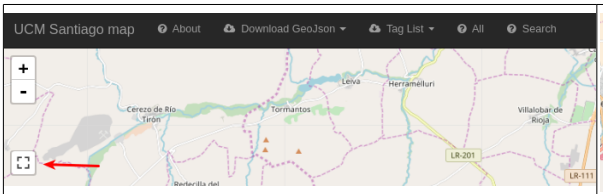


5. Ampliar en pantalla completa.

   Para una visualización más clara y completa podemos ampliar a pantalla completa nuestro mapa web.



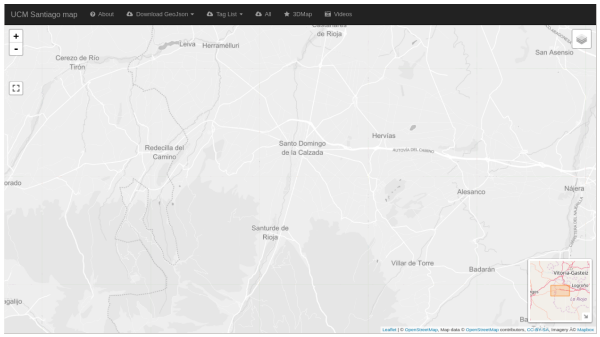
.- Visualización del mapa a pantalla completa, el menú de opciones desaparece.

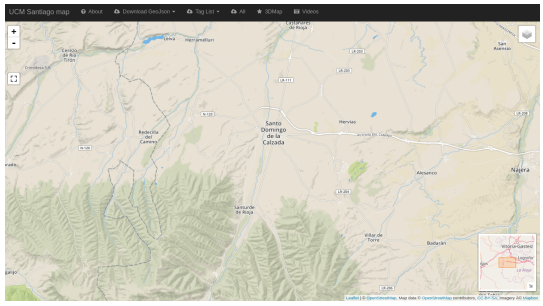
	
.- Ampliar a pantalla completa.	.- Salir del modo pantalla completa

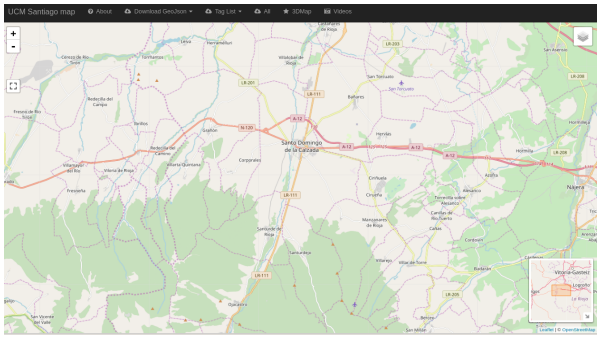
6. Mostrar un mapa base con diferentes estilos.

.- Capa donde se podrá elegir el estilo del mapa base.Se podrá elegir el estilo del mapa base entre las siguientes opciones procedentes de mapbox:

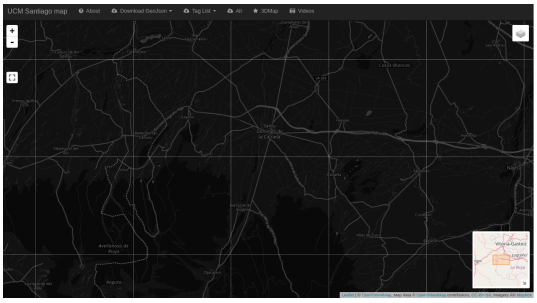
mapbox.light, mapbox.streets, mapbox.satellite-streets-v9, mapbox.dark



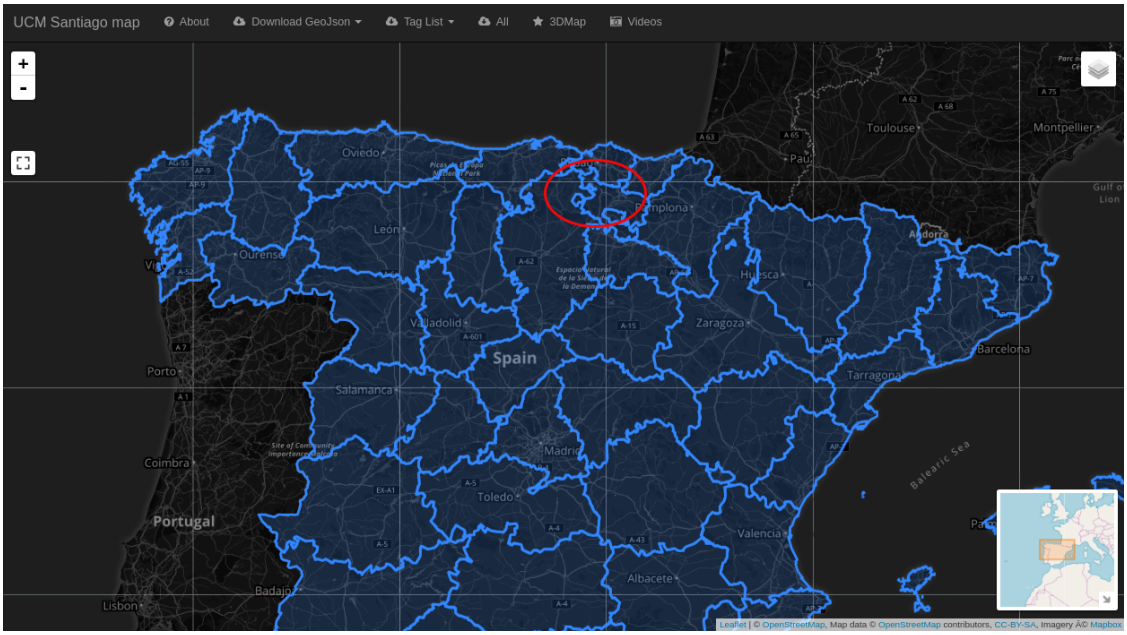




.- Style, mapbox.dark



.- Style, mapbox.satellite-streets-v9

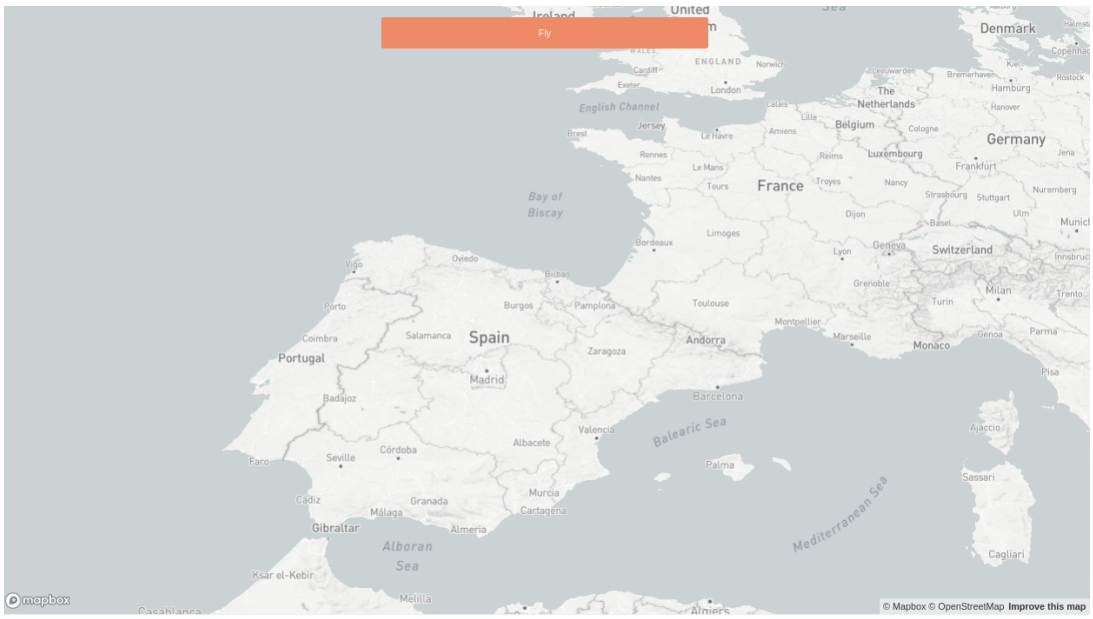


.- Mapa base style dark, con geoJson de provincias de España y zona de estudio.

7. Mapa en 3D.

   El objetivo de tener un mapa en 3D consistirá en poder cargar un zoom automatizado desde una perspectiva abierta a una más cerrada para localizar mejor el área donde se concentran la mayoría de nuestras fotografías de estudio.

   El botón “fly” activará el zoom hacia el área de estudio.



.- Mapa 3D.

8. Resumen de las funcionalidades y herramientas utilizadas en nuestro mapa en el acceso "about".

.- Capa modal con información resumen de nuestra aplicación.

7. - Demo y conclusiones

Una vez establecido el método de obtención de fotografías clasificadas de forma automatizada y habiendo hecho un análisis de los datos obtenidos, podemos considerar como conclusión principal que el proceso podría aplicarse a cualquier conjunto de imágenes con una relación entre ellas.

Es decir, la estructura, infraestructura y datos pueden servir a futuro para poder tener un sistema de clasificación que puede extenderse a otro tipo de información, en este caso hemos tomado fotografías relacionadas con el Camino de Santiago pero podría aplicar el mismo proceso a temas totalmente diferentes manteniendo el mismo sistema de clasificación.

Tanto la parte de procesamiento en Python, como la infraestructura de máquinas en Amazon y la renderización para la visualización de datos podrían mantenerse exactamente igual.

Hemos conseguido establecer un método para hacer crecer un SIG de datos fotográficos geolocalizados y clasificados automáticamente utilizando herramientas de programación.

La parte de visualización consideramos que ayuda mucho al proyecto, aportando la parte de facilidad de comprensión del proceso. Podrían realizarse mejoras para aumentar el volumen de datos que manejamos y sacar mayor número de clasificaciones posibles pero el volumen establecido en el proyecto ha sido suficiente para establecer que podría ser extensible.

Accesos directos por url a las funcionalidades desarrolladas en el proyecto.
Repositorio de código: https://github.com/rojorubi/ucm.sig.santiagomap **Página principal:** https://s3.us-east-2.amazonaws.com/ucmap/index.html **Videos:** https://s3.us-east-2.amazonaws.com/ucmap/videos.html **Mapa 3d:** https://s3.us-east-2.amazonaws.com/ucmap/3dmapExample.html#5/42.43/-2.96/-17.6/45

Bibliografía y páginas web de referencia

Google API Vision Cloud
https://cloud.google.com/vision/?hl=es

Flickr API
https://www.flickr.com/services/api/

Amazon Web Services (AWS) Documentation
https://aws.amazon.com/es/documentation/

Python references
https://www.python.org/

Leaflet - a JavaScript library for interactive maps
https://leafletjs.com/
http://leafletjs.com/reference.html
https://github.com/Leaflet/Leaflet.markercluster

Coordenadas de un lugar
https://www.coordenadas.com.es/

Proyectos interesantes
https://github.com/simonepri/geo-maps
https://github.com/datourbano/logrono
http://geojson.io